Nmap & SNORT

Eric Carestia

Dr. Janusz Zalewski

CNT 4104 Fall 2011 – Computer Networks

Florida Gulf Coast University

Ft. Myers, Florida

November 2011

1. Introduction

1.1 Project Overview

The purpose of this project is to utilize a port scanning application, NMap, to find open ports and/or potential vulnerabilities on another target PC. Concurrently, while the 'attacker' PC is scanning the target for vulnerabilities, the virtualized Intrusion Detection System (IDS) will be monitoring all inbound and outbound network traffic through a bridged network connection on the Hosting PC (Target). The IDS on the virtualized PC shall attempt to detect these scans with another application called SNORT with customized detection rules.

The Intrusion Detection System (IDS) looks for attack signatures, which are specific prototypes that usually indicate malicious or suspicious intent. This project shall demonstrate the use of SNORT, which is an open source network intrusion prevention and detection system utilizing a rule-driven language, to detect and react to ping attacks from the Attacker PC.

In the event of an intrusion attempt, the IDS will utilize a MySQL database which provides a mechanism to store and later query alerts that are generated by SNORT.

1.1.1 Project Resources

The software resources listed below were taken from their respective user installation manuals and are listed as required dependencies. The following resources were used in the creation of this project:

- Nmap software (http://nmap.org)
- Zenmap GUI frontend for Nmap (NMap)
- SNORT software (http://snort.org)
- Virtualization software- Oracle VirtualBox
- Snort-MySQL with all updated packages (SNORT IDS)

- Libpcap0.8-dev (SNORT IDS)
- Libmysqlclient15-dev (SNORT IDS)
- MySQL-client-5.0 (SNORT IDS)
- Bison (SNORT IDS)
- Flex (SNORT IDS)
- Apache2 Web Server Software (SNORT IDS)
- PHP5-gd (SNORT IDS)
- SSH (SNORT IDS and NMap)
- PHP-pear (SNORT IDS)

- A Computer with the following specifications is needed to conduct the experiment should have the following:
  - Intel Core i5/i7 Processor
  - 4GB of RAM
  - Windows XP Professional / Vista / 7 or Linux Operating System

1.1.2 Additional Resources

Documentation is available for download from the respective websites:

- Nmap [1] (Available online http://nmap.org/book)
- SNORT & accompanying documentation [2] (Available online http://www.snort.org/docs)

In addition, printed documentation exists which proved helpful at various stages of the project

1. Alder, Raven et.al. *SNORT 2.1 Intrusion Detection*. Second Edition. 2004 Syngress Publishing

2. Koziol, Jack. *Intrusion Detection with SNORT*. 2003, SAMS Publishing

2. Problem Description

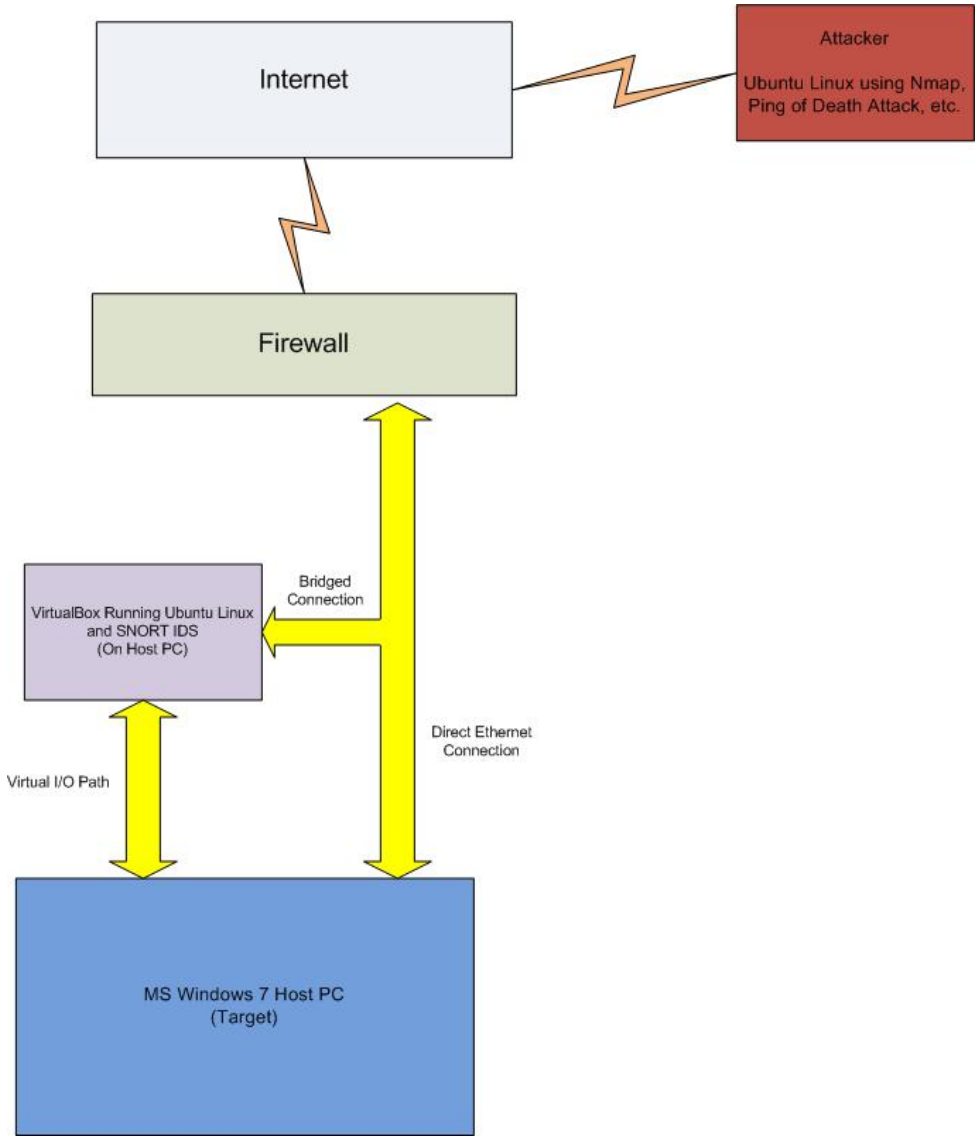
Internet
Attacker
Ubuntu Linux using Nmap, Ping of Death Attack, etc.
Firewall
VirtualBox Running Ubuntu Linux and SNORT IDS (On Host PC)
Bridged Connection
Direct Ethernet Connection
Virtual I/O Path
MS Windows 7 Host PC (Target)

Figure 2.1– Proposed System Architecture

2.1 Assumptions

This project simulates an attacker PC trying to scan a target PC over a virtualized network as shown in Figure 2.1:

- The Nmap software will run on the virtualized Attacker PC with an Internet connection.

- The Nmap software executes various port and ping scans against the Target PC and displays the results locally.
- The SNORT IDS will run on the Target PC in a virtualized environment (VirtualBox Virtual Machine).
- The SNORT IDS will listen to and capture inbound/outbound network traffic that it acquires from its bridged connection to the Host PC's physical Ethernet adapter.
- The SNORT IDS will utilize both pre-defined and user-defined rules to detect and report any intrusion attempt made by the Attacker PC.
- The SNORT IDS will log any intrusion attempt (e.g. port scans) made by the Attacker PC into a local MySQL database for later viewing.

2.2 Specific Requirements

The intended sequence of operations should follow these steps:

- Turn the attacker PC and target PC on, startup the virtualized IDS PC.
- Turn on all network interface hardware (Ethernet & Bridged Connection) and connect to the internet.
- Start the Nmap software (Attacker PC).
- Start the SNORT software (Virtualized IDS PC).
- The Attacker PC shall utilize the results obtained from nmap port scans to find a vulnerability in the Target PC's network.
- SNORT shall run on the target computer with an Internet connection.
- SNORT will act as a "middle-man" monitoring traffic between the host PC (Target) and the Internet.

- In the event an intrusion attempt is made, the intrusion attempt will be logged into a MySQL database for later viewing.

It is assumed that the Attacker uses:

- Zenmap display for Nmap (attacker).
- Various Nmap port and ping scans.

It is assumed that the SNORT IDS uses:

- Snort Intrusion Detection System v 2.9.1.2
- MySQL Client 5.
- Apache2 Webserver.

3. Solution Plan

3.1 Solution Plan Details

The plan for creating a solution to the problem described in Section 2.1 is a three phase plan of action, as follows:

Phase 1- Since there is no previous knowledge of how the SNORT & NMap software specifically functions, two textbooks [6][7] are needed to establish basic understanding. In addition, a basic demo program is to be implemented to verify that both SNORT & Nmap are working properly on both the Target and Attacker PC, respectively.

Phase 2- The Intrusion Detection System + Database log (SNORT) and the NMap scanner will be planned out using diagrams before any programs are developed/installed. Once this is finished, the development of customized detection rules for the Intrusion Detection System and customized port/vulnerability scans will begin. Once these rules / scans are developed, phase 3 will then begin.

Phase 3- After the customized detection rules are created and successfully loaded into the Target PC; and customized vulnerability scans are created for the Attacker PC, testing will begin. The testing plan will be developed and then each individual test that needs to be run to demonstrate proper functionality will be run in sequential order to ensure that both that SNORT and NMap along with their respective customizations are working properly. The test plan will be updated in the case that there are problems that arise during the testing phase. Phase 3 is discussed in more detail in Section 3.2 (Solution Design)

3.2 SNORT Solution Design – Target PC

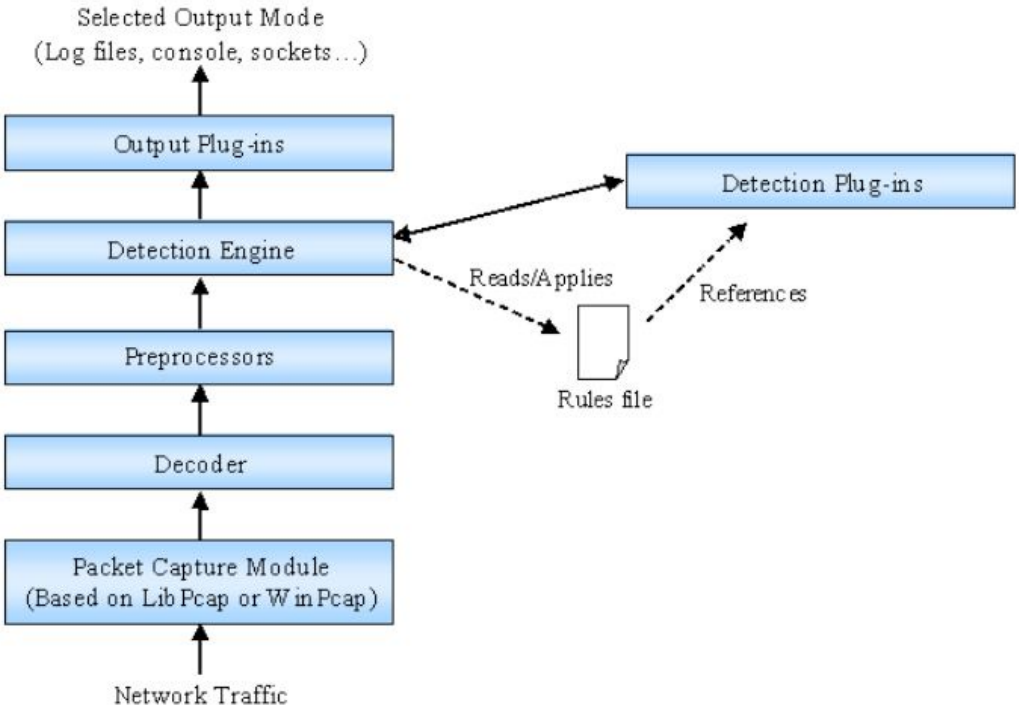

Figure 3-1 SNORT Dataflow Diagram [8]

SNORT utilizes a sequential methodology to capture, identify, and categorize network traffic on a LAN. In Figure 3.1, Network traffic is initially captured by the LIbPcap or WinPcap (as listed in Section 1.1.1 Required Resources) software libraries. These captured network packets are then passed along to a decoder which deciphers their contents. The decoded packets are then placed through a round of preprocessing with pre-established detection rules for the most common types of traffic, exploits, etc. (e.g. AOL Instant Messenger, Streaming Multimedia Traffic). After the preprocessing, the traffic packets are then sent through to the detection engine which then uses specific rule sets that will screen the traffic and either allows ('whitelists') or denies ('blacklists') the packets based upon the many available rule sets at its disposal. In the event of prohibited traffic or intrusion attempts, the output plugins (AcidBase, Snorby, etc.) will then write the event to a MySQL Database[3] shown in Figure 3-3 where the contents can be displayed to the user through various means (database queries, etc.).

3.2.1 SNORT Database Design

When suspicious or malicious activity occurs, generally there is not a single ‘event’; it can be several hundred or thousand ‘events’ that occur. SNORT can utilize a variety of ways to categorize and log intrusion attempts. For this project, MySQL was chosen as it is the most popular open source database software available. As per the user manual available from http://www.snort.org , the database that logs the intrusion attempts must contain 16 rows as shown in Figure 3.2:

```
| Tables_in_snort  |
+------------------+
| data             |
| detail           |
| encoding         |
| event            |
| icmphdr          |
| opt              |
| reference        |
| reference_system |
| schema           |
| sensor           |
| sig_class        |
| sig_reference    |
| signature        |
| tcphdr           |
| udphdr           |
+------------------+
```

Figure 3.2 MySQL database table “snort”

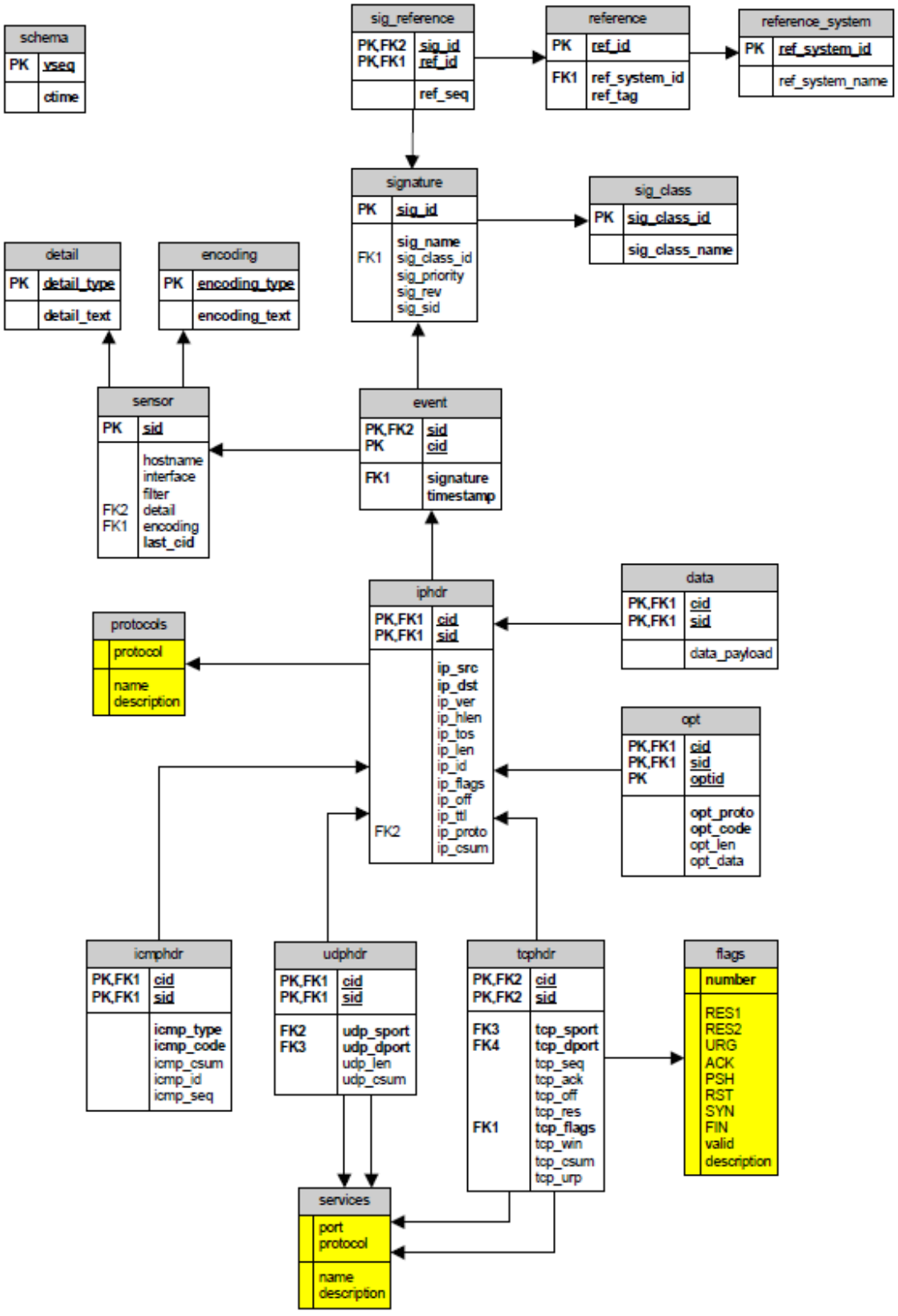

Figure 3.3 MySQL Database (Table name- snort) Relational Diagram

3.2.2 Definition of Each Table in Database Snort

This subsection explains the purpose of each row in table snort in Figure 3-2:

The “data” table contains the payload for each packet that triggers an alert.

The “detail” table contains information about how much detail is logged with a packet (default length- two rows)

The “encoding” table shows the types of encoding used when logging data packets. By default, it contains three types of logging: hex, base64, and ASCII.

The “event” table lists all events and stores a timestamp for these events.

The “icmphdr” table contains information about the ICMP header of packets that are logged into the database.

The “iphdr” table contains all fields of the IP header for logged data packets. The information includes source and destination IP addresses, IP protocol version, IP header length, type of service (TOS) value, time to live (TTL) value, and so on.

The “opt” table contains user defined options.

The “reference” and “reference_system” tables contain information about reference sites used to get more information about a specific vulnerability. This is the same information that is used inside the SNORT rule sets used to classify whether traffic is good or bad.

The “schema” table contains the version of the database schema.

The “sensor” table contains information about different sensors that are logging data to the SNORT database. If there is only one sensor (like in this project), this table will only contain one row.

The “sig_class” contains information about different classes of SNORT rules (attempted-recon, misc-attack, etc.).

The “signature” table contains information about signatures that generated the alerts.

The “tcphdr” table contains the information in the TCP header of a packet if the packet is of the TCP type.

The “udphdr” table contains the information in the UDP header of a packet if the packet is of the UDP type.

While there are 16 rows in this particular database, for this project we are only concerned with rows **signature** and **event** as they provide the type of intrusion attempt and what time the attempt was made. Other tables provide details that are beyond the scope of this project.

3.2.3 Creating a Custom SNORT Detection Rule

One of the best features of Snort is its rule engine and language. Snort's rule engine provides an extensive language that enables the users to write their own rules, allowing the user to extend it to meet the needs of a specific network.

A Snort rule can be broken down into two basic parts, the rule header and options for the rule. The rule header contains the action to perform, the protocol that the rule applies to, and the source and destination addresses and ports. The rule options allows users to create a descriptive message to associate with the rule, as well as check a variety of other packet attributes by making use of Snort's extensive library of plug-ins[7].

Here's the general form of a Snort rule:

action proto src_ip src_port direction dst_ip dst_port (options)

Where the specific parts mean the following:

action - this part specifies what action should be taken in the event of an intrusion attempt, this is generally set to “alert”.

proto - this part specifies the protocol to watch (e.g. TCP, IP, UDP, etc.)

src_ip- this part specifies the range of IP addresses to log traffic from.

src_port – this part specifies a port or port range to listen for intrusion attempts.

direction- this part specifies the direction of the checked network traffic. Three values are possible: inbound, outbound, or bidirectional.

dst_ip- this portion specifies the destination IP or range of destination IP addresses that the checked network traffic is intended for.

dst_port- this portion specifies the destination port number or range of port numbers that the checked network traffic is intended for.

(options)- this part provides extensive flexibility for the user to designate any other criteria for logging and categorizing captured network traffic.

When a packet comes in, its source and destination IP addresses and ports are then compared to the rules in the rule set. If any of them are applicable to the packet, then the options are compared to the packet. If all of these comparisons return a match, then the specified action is taken.

Snort provides several built-in actions that the user can use when crafting custom rules. To simply log the packet that matches a rule, use the **log** action. The **alert** action generates an alert using the method specified in your configuration file or on the command line, in addition to logging the packet. One nice feature is that the user can have very general rules and then create exceptions by writing a rule that uses the **pass** action. This works especially well when one is using the rules distributed with Snort, but are frequently getting false positives for some of them. If this happens and it's not a security risk to ignore them, one can simply write a **pass** rule for it.

The last two built-in rule actions are used together to dynamically modify Snort's rule set at runtime. These are the activate and dynamic actions. Rules that use the dynamic action are just like a log rule, except they will be considered only after they have been enabled by an **activate** rule. To accomplish this, Snort enforces the use of the **activates** and **activated_by** rule options in order to know what dynamic rules to enable

once an **activate** rule has been triggered. In addition, dynamic rules are required to specify a **count** option in order for Snort to limit how many packets the rule will record.

For instance, if the user wanted to start recording packets once an exploit of a SSH daemon on 192.168.1.21 was noticed, one could use a couple of rules similar to these:

**activate tcp any any -> 192.168.1.21 22 (content:"/bin/sh"; activates:1; **

msg:"Possible SSH buffer overflow";)

dynamic tcp any any -> 192.168.1.21 22 (activated_by:1; count:100;)

These two rules aren't completely foolproof, but if someone were to run an exploit with shell code against an SSH daemon, it would most likely send the string ***/bin/sh*** in the clear in order to spawn a shell on the system being attacked. In addition, since SSH is encrypted, strings like that wouldn't be sent to the daemon under normal circumstances. Once the first rule is triggered, it will activate the second one, which will record 100 packets and then stop. This is useful, since one might be able to catch the intruder downloading or installing a root kit within those first few packets and be able to analyze the compromised system much more quickly.

The user can also define custom rule actions, in addition to those that Snort has built-in. This is done with the **ruletype** keyword:

ruletype redalert

{

type alert

output alert_syslog: LOG_AUTH LOG_ALERT

output database: log, mysql, user=snort dbname=snort host=localhost

}

This custom rule **action** tells Snort that it behaves like the alert rule action, but specifies that the alerts should be sent to the syslog daemon, while the packet will be logged to a database. When defining a custom action, one can use any of Snort's output plug-ins, just as one would if one were configuring them as the primary output method.

Snort's detection engine supports several protocols. The proto field is used to specify what protocol the custom rule applies to. Valid values for this field are **ip, icmp**, **tcp**, and **udp**.

The other fields in a Snort rule are used to specify the source and destination IP addresses and ports of the packet, as well as the direction the packet is traveling. Snort can accept a single IP or a list of addresses. When specifying a list of IP address, one should separate each one with a comma and then enclose the list within square brackets, like this:

[192.168.1.1,192.168.1.45,10.1.1.24]

When doing this, one has to be careful not to use any whitespace. One can also specify ranges of IP addresses using CIDR notation, or even include CIDR ranges within lists. Snort also allows the user to apply the logical NOT operator (!) to an IP address or CIDR range to specify that the rule should match all but that address or range of addresses.

As with IP addresses, Snort can accept single ports as well as ranges. To specify a range, use a colon character to separate the lower bound from the upper bound. For example, if one wanted to specify all ports from 1 to 1024, one would do it like this:

1:1024

It is also apply the NOT operator to a port, thereby allowing the user to specify a range of ports without an upper or lower bound.

For instance, if one only wanted to examine ports greater than 1024, one would do it this way:

1024:

Similarly, one could specify ports less than 1024 by doing this:

:1024

If one does not care about the IP address or port, one can simply specify **any**.

The **direction** field is used to tell Snort which IP address and port is the source and which pair is the destination. In earlier versions of Snort one could use either -> or <- to specify the direction. However, the<- operator was removed since one can make either one equivalent to the other by just switching the IP addresses and port numbers. Snort does have another direction operator in addition to ->, though. Specifying <> as the direction tells Snort that one wants the rule to be applied bidirectional. This is especially useful when using log rules or dynamic rules, since one can log both sides of the TCP stream rather than just one direction.

The next part of the rule includes the options. This part lets one specify many other attributes to check against. Each option is implemented through a Snort plug-in. When a rule that specifies an option is triggered, Snort will run through the option's corresponding plug-in to perform the check against the packet. Snort has over 40 plug-ins—too many to cover in detail in this project. Here are some of the more useful ones.

The single most useful option is **msg**. This option allows one to specify a custom message that will be logged in the alert when a packet matching the rule is detected. Without it, most alerts wouldn't make much sense at first glance. This option takes a string enclosed in quotes as its argument.

For example, this specifies a logical message whenever Snort notices any traffic that is sent from 192.168.1.35:

alert tcp 192.168.1.35 any -> any any (msg:"Traffic from 192.168.1.35";)

Be sure not to include any escaped quotes within the string. Snort's parser is a simple one and does not support escaping characters.

Another useful option is content, which allows one to search a packet for a sequence of characters or hexadecimal values. If searching for a string, just put it in quotes. In addition, if

one wants it to do a case-insensitive search, add **nocase**; to the end of all the options. However, if one is looking for a sequence of hexadecimal digits, one must enclose them in | characters.

This rule will trigger when it sees the digit 0x90:

alert tcp any any -> any any (msg:"Possible exploit"; content:"|90|";)

This digit is the hexadecimal equivalent of the **NOP** instruction on the x86 architecture and is often seen in exploit code since it can be used to make buffer overflow exploits easier to write.

The **offset** and **depth** options can be used in conjunction with the content option to limit the searched portion of the data payload to a specific range of bytes.

If one wanted to limit content matches for **NOP** instructions to between bytes 40 and 75 of the data portion of a packet, one could modify the previously shown rule to look like this:

**alert tcp any any -> any any (msg:"Possible exploit"; content:"|90|"; **

offset:40; depth:75;)

One can also match against packets that do not contain the specified sequence by prefixing it with a **!**. In addition, many shell code payloads can be very large compared to the normal amount of data carried in a packet sent to a particular service. One can check the size of a packet's data payload by using the **dsize** option. This option takes a number as an argument. In addition, One can specify an upper bound by using the **<** operator, or can choose a lower bound by using the **>** operator. Upper and lower bounds can be expressed with **<>**.

For example:

**alert tcp any any -> any any (msg:"Possible exploit"; content:"|90|"; **

offset:40; depth:75; dsize: >6000;)

This modifies the previous rule to match only if the data payload's size is greater than 6000 bytes, in addition to the other options criteria.

To check the TCP flags of a packet, Snort provides the **flags** option. This option is especially useful for detecting port scans that employ various invalid flag combinations.

For example, this rule will detect when the SYN and FIN flags are set at the same time:

alert any any -> any any (flags: SF,12; msg: "Possible SYN FIN scan";)

Valid flags are **S** for SYN, **F** for FIN, **R** for RST, **P** for PSH, **A** for ACK, and **U** for URG. In addition, Snort lets one check the values of the two reserved flag bits. It is possible to specify these by using either 1 or 2. One can also match packets that have no flags set by using 0. There are also several operators that the flags option will accept. One can prepend either a + , *, or ! to the flags, to match on all the flags plus any others, any of the flags, or only if none of the flags are set, respectively.

One of the best features of Snort is that it provides many plug-ins that can be used in the **options** field of a rule. The options discussed here should get oneself off to a good start. However, if one would want to write more complex rules, consult Snort's excellent rule documentation, which contains full descriptions and examples for each of Snort's rule options. The Snort User's Manual is available at http://www.snort.org/docs/writing_rules/.

3.4 NMap Solution Plan – Attacker PC

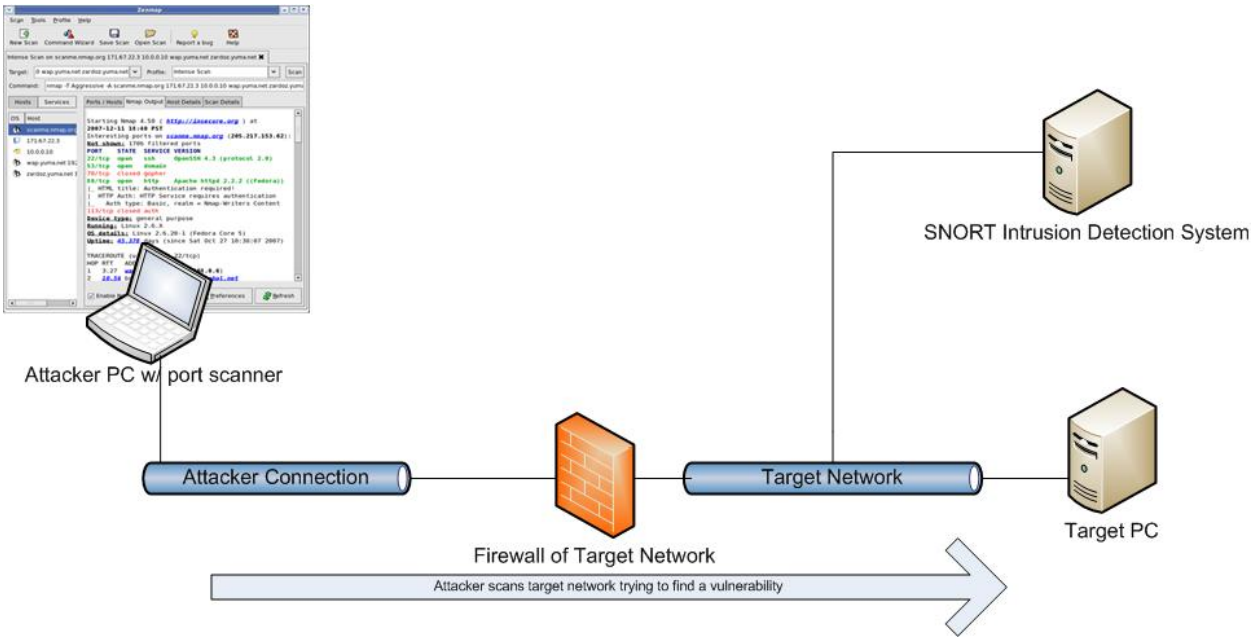
SNORT Intrusion Detection System
Attacker PC w/ port scanner
Attacker Connection
Target Network
Target PC
Firewall of Target Network
Attacker scans target network trying to find a vulnerability

Figure 3-3 Nmap Dataflow Diagram

Nmap ("Network Mapper") is an open source tool for network exploration and security auditing. It was designed to rapidly scan large networks, although it works fine against single hosts. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. While Nmap is commonly used for security audits, many systems and network administrators find it useful for routine tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime[1].

The output from Nmap is a list of scanned targets, with supplemental information on each depending on the options used. Key among that information is the "interesting ports table". That table lists the port number and protocol, service name, and state. The state is open, filtered, closed, or unfiltered. Open means that an application on the target machine is listening for connections/packets on that port. Filtered means that a firewall, filter, or other network obstacle is blocking the port so that Nmap cannot tell whether it

is open or closed. Closed ports have no application listening on them, though they could open up at any time. Ports are classified as unfiltered when they are responsive to Nmap's probes, but Nmap cannot determine whether they are open or closed. Nmap reports the state combinations **open|filtered** and **closed|filtered** when it cannot determine which of the two states describe a port[1]. The port table may also include software version details when version detection has been requested. When an IP protocol scan is requested (**-sO**), Nmap provides information on supported IP protocols rather than listening ports.

In addition to the interesting ports table, Nmap can provide further information on targets, including reverse DNS names, operating system guesses, device types, and MAC addresses.

3.4.1 Nmap Targeting Specifications

Everything on the Nmap command-line that isn't an option (or option argument) is treated as a target host specification. The simplest case is to specify a target IP address or hostname for scanning.

Sometimes the user may wish to scan a whole network of adjacent hosts. For this, Nmap supports CIDR-style addressing. The user can append **/<numbits>** to an IPv4 address or hostname and Nmap will scan every IP address for which the first **<numbits>** are the same as for the reference IP or hostname given. For example, 192.168.10.0/24 would scan the 256 hosts between 192.168.10.0 (binary: 11000000 10101000 00001010 00000000) and 192.168.10.255 (binary: 11000000 10101000 00001010 11111111), inclusive. 192.168.10.40/24 would scan exactly the same targets. Given that the host scanme.nmap.org is at the IP address 64.13.134.52, the specification scanme.nmap.org/16 would scan the 65,536 IP addresses between 64.13.0.0 and 64.13.255.255. The smallest allowed value is /0, which targets the whole Internet. The largest value is /32, which scans just the named host or IP address because all address bits are fixed[3].

CIDR notation is short but not always flexible enough. For example, the user might want to scan 192.168.0.0/16 but skip any IPs ending with .0 or .255 because they may be used as subnet

network and broadcast addresses. Nmap supports this through octet range addressing. Rather than specify a normal IP address, the user can specify a comma-separated list of numbers or ranges for each octet. For example, 192.168.0-255.1-254 will skip all addresses in the range that end in .0 or .255, and 192.168.3-5,7.1 will scan the four addresses 192.168.3.1, 192.168.4.1, 192.168.5.1, and 192.168.7.1. Either side of a range may be omitted; the default values are 0 on the left and 255 on the right. Using **-** by itself is the same as 0-255, but remember to use 0- in the first octet so the target specification doesn't look like a command-line option. Ranges need not be limited to the final octets: the specifier 0-255.0-255.13.37 will perform an Internet-wide scan for all IP addresses ending in 13.37. This sort of broad sampling can be useful for Internet surveys and research[1].

IPv6 addresses can only be specified by their fully qualified IPv6 address or hostname. Please note that CIDR and octet ranges aren't yet supported for IPv6.

3.4.2 Port Scanning Basics

While Nmap has grown in functionality over the years, it began as an efficient port scanner, and that remains its core function. The simple command **nmap *<target>*** scans 1,000 TCP ports on the host ***<target>*.** While many port scanners have traditionally lumped all ports into the open or closed states, Nmap is much more granular. It divides ports into six states: **open**, **closed**, **filtered**, **unfiltered**, **open|filtered**, or **closed|filtered** [1].

These states are not intrinsic properties of the port itself, but describe how Nmap sees them. For example, an Nmap scan from the same network as the target may show port 135/tcp as open, while a scan at the same time with the same options from across the Internet might show that port as filtered. There are six (6) port states recognized by NMap, as follows:

open

> An application is actively accepting TCP connections, UDP datagrams or SCTP associations on this port. Finding these is often the primary goal of port scanning.

Security-minded people know that each open port is an avenue for attack. Attackers and pen-testers want to exploit the open ports, while administrators try to close or protect them with firewalls without thwarting legitimate users. Open ports are also interesting for non-security scans because they show services available for use on the network.

closed

A closed port is accessible (it receives and responds to Nmap probe packets), but there is no application listening on it. They can be helpful in showing that a host is up on an IP address (host discovery, or ping scanning), and as part of OS detection. Because closed ports are reachable, it may be worth scanning later in case some open up. Administrators may want to consider blocking such ports with a firewall.

filtered

Nmap cannot determine whether the port is open because packet filtering prevents its probes from reaching the port. The filtering could be from a dedicated firewall device, router rules, or host-based firewall software. These ports frustrate attackers because they provide so little information. Sometimes they respond with ICMP error messages such as type 3 code 13 (destination unreachable: communication administratively prohibited), but filters that simply drop probes without responding are far more common. This forces Nmap to retry several times just in case the probe was dropped due to network congestion rather than filtering. This slows down the scan dramatically.

unfiltered

The unfiltered state means that a port is accessible, but Nmap is unable to determine whether it is open or closed. Only the ACK scan, which is used to map firewall rule sets, classifies ports into this state. Scanning unfiltered ports with other scan types such as Window scan, SYN scan, or FIN scan, may help resolve whether the port is open.

open|filtered

Nmap places ports in this state when it is unable to determine whether a port is open or filtered. This occurs for scan types in which open ports give no response. The lack of response could also mean that a packet filter dropped the probe or any response it elicited. So Nmap does not know for sure whether the port is open or being filtered. The UDP, IP protocol, FIN, NULL, and Xmas scans classify ports this way.

closed|filtered

This state is used when Nmap is unable to determine whether a port is closed or filtered. It is only used for the IP ID idle scan.

3.4.3 Creating a Port/Vulnerability Scan for Nmap

Here are some Nmap usage examples, from the simple and routine to a little more complex and esoteric. Some actual IP addresses and domain names are used to make things more concrete. In their place you should substitute addresses/names from *your own network*. While I don't think port scanning other networks is illegal, some network administrators don't appreciate unsolicited scanning of their networks and may complain. Getting permission first is the best approach. For a full list of Nmap options and command switches, please see www./nmap.org/book for more detailed information .

For testing purposes, you have permission to scan the host scanme.nmap.org. This permission only includes scanning via Nmap and not testing exploits or denial of service attacks. Be polite! Please do not initiate more than a dozen scans against that host per day. If this free scanning target service is abused, it will be taken down and Nmap will report Failed to resolve given hostname/IP: scanme.nmap.org. These permissions also apply to the hosts scanme2.nmap.org, scanme3.nmap.org, and so on, though those hosts do not currently exist.

nmap -v scanme.nmap.org

This option scans all reserved TCP ports on the machine scanme.nmap.org . The -v option enables verbose mode.

nmap -sS -O scanme.nmap.org/24

Launches a stealth SYN scan against each machine that is up out of the 256 IPs on the class C sized network where Scanme resides. It also tries to determine what operating system is running on each host that is up and running. This requires root privileges because of the SYN scan and OS detection.

nmap -sV -p 22,53,110,143,4564 198.116.0-255.1-127

Launches host enumeration and a TCP scan at the first half of each of the 255 possible eight-bit subnets in the 198.116 class B address space. This tests whether the systems run SSH, DNS, POP3, or IMAP on their standard ports, or anything on port 4564. For any of these ports found open, version detection is used to determine what application is running.

nmap -v -iR 100000 -Pn -p 80

Asks Nmap to choose 100,000 hosts at random and scan them for web servers (port 80). Host enumeration is disabled with -Pn since first sending a couple probes to determine whether a host is up is wasteful when only probing one port on each target host anyway.

nmap -Pn -p80 -oX logs/pb-port80scan.xml -oG logs/pb-port80scan.gnmap 216.163.128.20/20

This scans 4096 IPs for any web servers (without pinging them) and saves the output in grep-able and XML formats.

3.5 Test Plan

3.5.1 Description of Test Environment

The test environment includes two (2) virtualized PC's running Ubuntu Linux inside a Host PC running Windows 7. The Virtual PC's are loaded before any testing can begin and all software are initialized and run before testing can begin to ensure all components are in proper working order.

3.5.2 Stopping Criteria

If an error occurs during any test case, the testing phase continues until either the SNORT IDS or Nmap port scanner performs correctly. The subsequent result of the test is then be recorded and documented accordingly. The test result documentation is updated for every iteration for the test phase.

If an error occurs in either the SNORT IDS or the Nmap port scanner's operation, the above is executed and returns to development or planning phases in order to ensure that the custom detection rules (SNORT IDS) or custom port scans (Nmap) operate according to specifications.

The system will be deemed complete when each and all test phases can be completed with zero errors. Once this happens, the documentation will be updated with the final test results and the system can be demonstrated.

3.5.3 Description of Individual Test Cases

Test Cases for SNORT IDS Target PC

Test Case IDS 1

Test Objective – The SNORT IDS is compiled and installed properly.

Test Description – Download SNORT IDS source code along with all dependencies, follow instructions given to successfully compile SNORT on the Target PC.

Test Conditions- this test should be completed before any other test cases start as this is a vital part of the testing phase.

Expected Results- Once SNORT's source code and dependencies are downloaded and compiled, the SNORT IDS should initialize and execute.

Test Case IDS 2

Test Objective – SNORT IDS detects traffic with default rule sets.

Test Description- From a command window, start SNORT in the Intrusion Detection Mode.

Test Conditions- This test should be completed before any other test cases start as this is also a vital part of the testing phase.

Expected Results- Once SNORT is executed in intrusion detection mode, SNORT will start successfully and signal it is capturing packets.

Test Case IDS 3

Test Objective- SNORT IDS detects Nmap TCP scans

Test Description- After the Attacker PC runs a port/vulnerability scan on the target, all scanning activity performed will be logged into the MySQL SNORT database on the Target PC

Test Conditions - This test should be executed to ensure that the default rule sets are functioning properly and the MySQL SNORT database is properly setup and configured.

Expected Results- The scans will be logged into the SNORT database, where it can be queried to verify detection.

Test Case IDS 4

Test Objective- Verify results of SNORT IDS logging Nmap scans to MySQL Database

Test Description- The user reviews the results of the query executed in Test Case IDS to ensure entries into the database are a result of the Attacker PC executing port scans by viewing the data stored in the "signature" portion of the query results.

Expected Results- The query results should show entries with signatures that correspond to NMap scans against the Target PC as shown in Figure 4-2.

Test Cases for Nmap Attacker PC

Test Case NMap 1

Test Objective- Nmap + Zenmap GUI frontend is installed correctly.

Test Description- Download Nmap + Zenmap GUI frontend and install from the command line with **sudo apt-get install xxxxx**

Test Conditions- This test must be completed because it is a vital part of the overall project and is necessary for the Attacker PC to perform it's function.

Expected Results- Nmap + Zenmap GUI download, install and startup properly.

Test Case NMap 2

Test Objective – Use Nmap to scan the Target PC in an attempt to find open ports/vulnerabilities.

Test Description – Upon scanning the results from running the Nmap execution, the list of open/filtered/closed ports should be visible.

Test Conditions- The completion of this test cannot be guaranteed because no ports may be open on a real target, but for this project port 80 and 3389 will be opened on the target.

Expected Results- The Nmap Scan will display open ports of 80 and 3389 on the Target PC.

4. Implementation and Results

SNORT and Nmap were successfully downloaded, compiled, installed and executed as described in section 3. The SNORT rule is fairly intuitive and facilitated for fast rule creation. As such, I created a custom scan rule set to supplement the default scan.rules rule that is quite lacking. The source code for "custom_scan.rules" is in the following section 4.1

Nmap is very flexible and provides nearly infinite permutations and combinations of scans that can be performed. After some careful research and executing example scans via nmap.org and the predefined scans in the Zenmap GUI, I was able to create a scan that would successfully scan the Target PC and display the open ports. The custom command(s) are listed in section 4.2

4.1 Custom SNORT Rule Set- custom_scan.rules

This file was custom made for the express purpose of extending the rather small scan.rules rule set that is preloaded with the default SNORT IDS installation. This file expands to include many other port scanners not limited to Nmap. This custom_scan.rules can be added to the scan.rules file that is included with SNORT.

 # $Id:custom_scan.rules, vrtbuild Exp $

#----------

 # SCAN RULES

#----------

 # These signatures are representative of network scanners. These include

 # port scanning, ip mapping, and various application scanners.

#

#

 # alert tcp $EXTERNAL_NET 10101 -> $HOME_NET any (msg:"SCAN myscan"; flow:stateless; ack:0; flags:S; ttl:>220; reference:arachnids,439; classtype:attempted-recon; sid:613; rev:7;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 113 (msg:"SCAN ident version request"; flow:to_server,established; content:"VERSION|0A|"; depth:16; reference:arachnids,303; classtype:attempted-recon; sid:616; rev:4;)

 # alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"SCAN cybercop os probe"; flow:stateless; dsize:0; flags:SF12; reference:arachnids,146; classtype:attempted-recon; sid:619; rev:7;)

 # alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN ipEye SYN scan"; flow:stateless; flags:S; seq:1958810375; reference:arachnids,236; classtype:attempted-recon; sid:622; rev:8;)

 # alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN synscan portscan"; flow:stateless; flags:SF; id:39426; reference:arachnids,441; classtype:attempted-recon; sid:630; rev:7;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN cybercop os PA12 attempt"; flow:stateless; flags:PA12; content:"AAAAAAAAAAAAAAAA"; depth:16; reference:arachnids,149; classtype:attempted-recon; sid:626; rev:8;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN cybercop os SFU12 probe"; flow:stateless; ack:0; flags:SFU12; content:"AAAAAAAAAAAAAAAA"; depth:16; reference:arachnids,150; classtype:attempted-recon; sid:627; rev:8;)

alert udp $EXTERNAL_NET any -> $HOME_NET 10080:10081 (msg:"SCAN Amanda client-version request"; flow:to_server; content:"Amanda"; fast_pattern:only; classtype:attempted-recon; sid:634; rev:5;)

alert udp $EXTERNAL_NET any -> $HOME_NET 49 (msg:"SCAN XTACACS logout"; flow:to_server; content:"|80 07 00 00 07 00 00 04 00 00 00 00 00|"; fast_pattern:only; reference:arachnids,408; classtype:bad-unknown; sid:635; rev:5;)

alert udp $EXTERNAL_NET any -> $HOME_NET 7 (msg:"SCAN cybercop udp bomb"; flow:to_server; content:"cybercop"; fast_pattern:only; reference:arachnids,363; classtype:bad-unknown; sid:636; rev:3;)

alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN Webtrends Scanner UDP Probe"; flow:to_server; content:"|0A|help|0A|quite|0A|"; fast_pattern:only; reference:arachnids,308; reference:url,www.netiq.com/products/vsm/default.asp; classtype:attempted-recon; sid:637; rev:7;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"SCAN SSH Version map attempt"; flow:to_server,established; content:"Version_Mapper"; fast_pattern:only; classtype:network-scan; sid:1638; rev:6;)

 # alert udp $EXTERNAL_NET any -> $HOME_NET 1900 (msg:"SCAN UPnP service discover attempt"; flow:to_server; content:"M-SEARCH "; depth:9; content:"ssdp|3A|discover"; fast_pattern:only; classtype:network-scan; sid:1917; rev:9;)

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN SolarWinds IP scan attempt"; icode:0; itype:8; content:"SolarWinds.Net"; fast_pattern:only; classtype:network-scan; sid:1918; rev:7;)

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SCAN cybercop os probe"; flow:stateless; ack:0; flags:SFP; content:"AAAAAAAAAAAAAAA"; depth:16; reference:arachnids,145; classtype:attempted-recon; sid:1133; rev:12;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 5000 (msg:"SCAN UPnP service discover attempt"; flow:to_server,established; content:"M-SEARCH "; depth:9; content:"ssdp|3A|discover"; fast_pattern:only; classtype:network-scan; sid:8081; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"SCAN Proxyfire.net anonymous proxy scan"; flow:to_server,established; content:"proxyfire.net/fastenv"; nocase; http_uri; reference:url,www.proxyfire.net/index.php; classtype:network-scan; sid:18179; rev:2;)

4.2 Nmap Custom Scans

Nmap give the user the incredible flexibility to scan a single computer to an entire subnet/network in nearly innumerable ways. This section will detail the scans used and created for this project.

TCP scan, all ports:

This scan will scan all TCP ports in the range 1-65535 on the host (shown here by 192.168.2.1)

nmap -p 1-65535 -T4 -A -v 192.168.2.1

-A switch allows Nmap to detect the target machine's OS.

-v switch specifies “verbose” mode, can be doubled (e.g. –vv) for even more verbose output.

TCP Scan, no ping:

Sometimes, an intrusion detection system is configured to detect ping scans of open ports, this particular scan does not ping the target, and essentially this scan lets one ‘fly under the radar’.

nmap -T4 -A -v -Pn 192.168.2.1

TCP Scan, plus UDP ports

nmap -sS -sU -T4 -A -v 192.168.2.1

This scan includes the previous TCP port scan, but adds scanning of the UDP ports too.

Quick Traceroute:

nmap -sn --traceroute 192.168.2.1

This scan will complete a “tracert” from the Attacker to the Target and display all network hops from source to target.

Quick scan “plus”

nmap -sV -T4 -O -F --version-light 192.168.2.1

This scan fragments the packets and detects the version of a service running on any open port on the target

Note: -F switch scans only those ports listed in the nmap_services file, which is much faster than scanning all 65535 ports.

Slow Comprehensive scan:

nmap -sS -sU –sO -T4 -A -v -PE -PP -PS80,443 -PA3389 -PU40125 -PY -g 53 --script "default or (discovery and safe)" 192.168.2.1

Note: This scan runs through all basic options (in order) and can take 12+ hours to complete!

-sS : stealth scan

-sU: scan UDP

-sO: IP Protocol Scan

-T4: IPv4

-v: verbose mode

-PE : Standard ICMP Echo Request

-PP: ICMP Timestamp Request

-PS: send SYN packets instead of ACK packets to target

4.3 Test Results

Currently, the code and software has been written for both the SNORT IDS and the Nmap executable and their respective hosting PCs. The Attacker PC can scan the Target PC and detect ports 80 and 3389 as open, concurrently, the Nmap scans are logged into the Snort Database on the Target PC demonstrating that the SNORT IDS and custom rules work as intended. This project can be executed as designed. See Section 4.3.1 and 4.3.2 for test results.

4.3.1 Results of Test Cases For SNORT

Test Case IDS 1 Results

Test Objective – The SNORT IDS is compiled and installed properly.

Test Description – Download SNORT IDS source code along with all dependencies, follow instructions given to successfully compile SNORT on the Target PC.

Test Conditions- this test should be completed before any other test cases start as this is a vital part of the testing phase.

Expected Results- Once SNORT's source code and dependencies are downloaded and compiled, the SNORT IDS should initialize and execute.

Actual Results- SNORT Software and all dependencies download, install and/or compile correctly without error. SNORT is initialized and is executed in packet detection mode See Figure 4-1 for results.

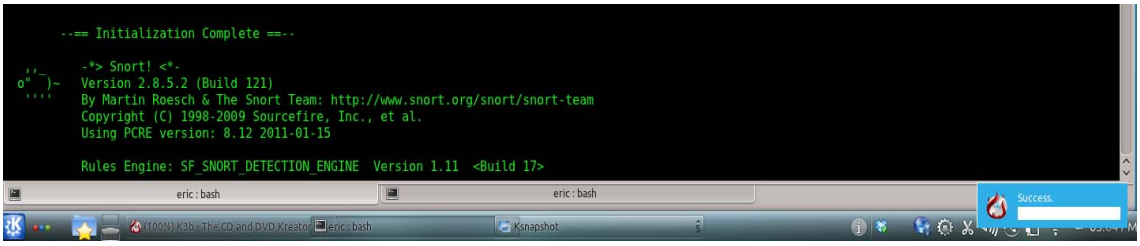

Figure 4-1 SNORT IDS successfully initialized and started

Test Case IDS 2 Results

Test Objective – SNORT IDS detects traffic with default rule sets.

Test Description- From a command window, start SNORT in the Intrusion Detection Mode.

Test Conditions- This test should be completed before any other test cases start as this is also a vital part of the testing phase.

Expected Results- Once SNORT is executed in intrusion detection mode, SNORT will start successfully and signal it is capturing packets.

Actual Results- SNORT IDS starts in intrusion detection mode, SNORT signals it is capturing packets.

Test Case IDS 3 Results

Test Objective- SNORT IDS detects Nmap TCP scans

Test Description- After the Attacker PC runs a port/vulnerability scan on the target, all scanning activity performed will be logged into the MySQL SNORT database on the Target PC

Test Conditions - This test should be executed to ensure that the default rule sets are functioning properly and the MySQL SNORT database is properly setup and configured.

Expected Results- The scans will be logged into the SNORT database, where it can be queried to verify detection.

Actual Results- The SNORT database is queried directly with the following query: *SELECT * FROM signature, event;* and the database that was previously empty, now displays the Nmap scans in their various forms along with a timestamp of their arrival and subsequent logging to the database as shown in Figure 4-2.

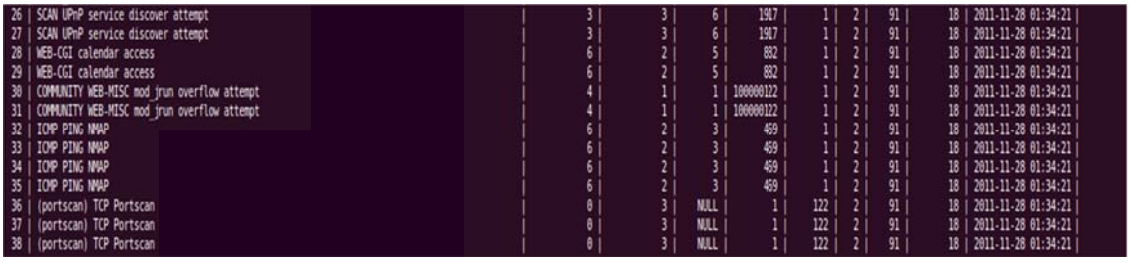

Figure 4-2 Results of NMap port scans logged by SNORT IDS into a MySQL Database

Test Case IDS 4 Results

Test Objective- Verify results of SNORT IDS logging Nmap scans to MySQL Database

Test Description- The user reviews the results of the query executed in Test Case IDS to ensure entries into the database are a result of the Attacker PC executing port scans by viewing the data stored in the "signature" portion of the query results.

Expected Results- The query results should show entries with signatures that correspond to NMap scans against the Target PC as shown in Figure 4-2.

Actual Results- The query on the table "snort" returns results that indicate that various TCP, UDP, and ICMP port scans were performed against the Target by the values shown in the "signature" field of the query result. The "event" field of the query result denotes what time was the port scan detected and subsequently logged into the database.

The fields we are concerned with are the first, second, and tenth (Signature ID, Signature Name, and Event Timestamp, respectively). Example query result as shown in bold from Figure 4.2. :

32 | ICMP PING NMAP| 6| 1| 469| 1| 2| 91| 18|2011-11-28 01:34:21|

The first field, **32**, denotes the signature ID number corresponding to a specific intrusion attempt.

The second field, **ICMP PING NMAP**, denotes the name corresponding to the value in the first field; in this case it is a specific type of NMap ping scan.

The tenth field, **2011-11-28 01:34:21**, denotes the date and time when the event was created.

4.3.2 Results of Test Cases for Nmap Attacker PC

Test Case NMap 1 Results

Test Objective- Nmap + Zenmap GUI frontend is installed correctly.

Test Description- Download Nmap + Zenmap GUI frontend and install from the command line with **sudo apt-get install xxxxx**

Test Conditions- This test must be completed because it is a vital part of the overall project and is necessary for the Attacker PC to perform its function.

Expected Results- Nmap + Zenmap GUI will download, install and startup properly.

Actual Results- Nmap + Zenmap download, install and execute as described.

Test Case NMap 2 Results

Test Objective – Use Nmap to scan the Target PC in an attempt to find open ports/vulnerabilities.

Test Description – Upon scanning the results from running the Nmap execution, the list of open/filtered/closed ports should be visible.

Test Conditions- The completion of this test cannot be guaranteed because no ports may be open on a real target, but for this project port 80 and 3389 will be opened on the target.

Expected Results- The Nmap Scan will display open ports of 80 and 3389 on the Target PC.

Actual Results- The Nmap scan does display ports 80 and 3389 as “open” on the Target PC as shown in figure 4-3.

5. Conclusion

For all iterations of testing, each case was tested to ensure the correct results, the SNORT IDS system works as described and successfully logs each intrusion attempt made by the Attacker PC; The Attacker PC successfully scans the Target PC and displays the results of having the two ports (80 & 3389) as open as described above. The information on the SNORT IDS for the Target PC was logged to the MySQL database, and the scan results are successfully displayed on the Attacker PC.

This project was an excellent primer into the world of intrusion detection systems, but it was not without its challenges. Initially, the SNORT IDS was installed on Ubuntu 11.04 and ran without incident. Upon upgrading to the newest version of Ubuntu, version 11.10, SNORT no longer worked. Uninstalling and then re-installing via the command line package installer proved useless. A workaround finally was found by not using the package installer from the command line, but compiling the software directly from the source code. This significantly increased delays as numerous required packages, libraries, and supporting software had to be downloaded, installed, and configured in order to bring SNORT back to working order.

The next step was to apply the knowledge gained from example SNORT rules to create my own set of rules that logged other types of port scanners to further flesh out the rather paltry default rule file for port scanners. With plenty of examples and the two textbooks, this proved to be straightforward.

The project's capabilities so far represent "the tip of the iceberg". Future projects have great potential for growth and expansion. One possible expansion is the implementation of a GUI Webpage front end for displaying the database results instead of querying the MySQL database and pouring through the tens of thousands of entries. Another possible expansion of SNORT is it's role as an intrusion *prevention* system. Numerous tutorials and example projects are available online at http://snort.org.

6. References

1. Nmap User Documentation. Available online http://nmap.org/book/man.html.

3. SNORT User Documentation. Available online http://www.snort.org/docs.

4. Zenmap GUI for Nmap. Available online http://nmap.org/zenmap/.

5. Oracle VirtualBox User Manual. Available Online http://dlc.sun.com.edgesuite.net/virtualbox/4.1.4/UserManual.pdf

6. The Apache Web Server Installation Guide. Available Online http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch20_:_The_Apache_Web_Server#Download_and_Install_The_Apache_Package

7. Alder, Raven et.al. *SNORT 2.1 Intrusion Detection*. Second Edition. 2004 Syngress Publishing

8. Koziol, Jack. *Intrusion Detection with SNORT*. 2003, SAMS Publishing

9. Karr, Harold. *Snort IDS, IPS, NSM, Hacking and…Beyond.* May 2009. Available online: http://harrykar.blogspot.com/2009/05/snort-ids-ips-nsm-andbeyond.html

Appendix – User Manual for SNORT + NMap Installation

Part A- SNORT Installation Instructions (Virtual IDS & Target PC)

A.1 Installation Overview

The following instructions enable a user to set up the virtualized PC with VirtualBox, install Ubuntu Linux, install SNORT, AcidBase GUI, and all related packages / software.

Please Note: The instructions in the following subsections are to be done on the Target PC and inside the virtualized IDS PC

All commands are in **boldface.**

A.2 Setup VirtualBox and Install Ubuntu Linux

Within Virtual Box for the IDS, the following settings are recommended:

- 512 MB RAM
- 12 GB Disk
- Eth0 host only network
- Eth1 bridged to host interface
- No sound device, printer or accessories

First, download VirtualBox from https://www.virtualbox.org/wiki/Downloads. Run the setup executable and follow the on screen instructions to complete the installation. Detailed instructions may be found: http://dlc.sun.com.edgesuite.net/virtualbox/4.1.4/UserManual.pdf

Second, download Ubuntu from http://www.ubuntu.com. Once the operating system is installed from defaults, several packages will have to be installed to support the Snort installation and its supporting applications.

I created bubba as my non-root user and bubba's home directory is referred to throughout this paper. If another user account is used, simply replace bubba with the appropriate user name.

Once the initial system has loaded and rebooted, in the GUI go to System > Administration > Synaptic Package Manager. You may have to click the reload button to get an updated list of packages. Inside the interface, choose the following applications for installation:

- libpcap0.8-dev
- libmysqlclient15-dev
- mysql-client-5.0
- mysql-server-5.0
- bison
- flex
- apache2
- libapache2-mod-php5
- php5-gd
- php5-mysql
- libtool
- libpcre3-dev
- php-pear
- vim
- ssh

Before installing Ubuntu, please note that Ubuntu prefers to have users execute commands requiring root level privileges by typing sudo in front of the command in question. I believe this to be a pain in the anatomy, so I set a password for the root user by typing:

sudo passwd

If you would prefer to use sudo and not set up a root level password, please feel free to do so. Most of the CLI commands in here were typed as root, so they will all need sudo in front of them and some of them won't work properly.

Before proceeding with the next section, download libnet-1.0.2a.tar.gz from *http://www.filewatcher.com/m/libnet-1.0.2a.tar.gz.140191.0.0.html.*

A.3 Get Snort

Go to http://snort.org and download the latest version of SNORT (tar.gz). If you have a registered account with snort.org, you can get more up to date rules. The Subscription Release provides registered users of Snort.org with immediate access to the most up to date Sourcefire VRT Certified Rules available. Real-time access requires a paid, annual subscription. For more information on subscriptions, please see:

http://www.snort.org/snort-rules/#rules.

A.4 Finish System Set-up and Compile Snort

Now we will finish the system set up by installing libnet and pcre. For more information on libnet, please see *http://libnet.sourceforge.net/.* Take the following steps:

- **cd /usr/local**
- **tar zxvf /home/bubba/Desktop/libnet-1.0.2a.tar.gz**
- **cd Libnet-1.0.2a**
- **./configure && make && make install**

In short, it's a shortcut to compile and install the libnet code extensions. If you'd like more information on configure, make and make install, please refer to *http://www.codecoffee.com/tipsforlinux/articles/27.html.*

To install Snort, execute the following commands:

- **cd /usr/local**
- **tar zxvf /home/bubba/Desktop/snort-2.8.4.1.tar.gz**
- **cd snort-2.8.4.1**
- **./configure --enable-targetbased && make && make install**

A.5 Set up Snort Environment

There are a few steps that need to take place in order to have snort run properly, mostly setting up some directories, getting the snort rules, moving some files around and creating the snort user. Execute the following:

- **mkdir /etc/snort**
- **mkdir /var/log/snort**
- **cd /etc/snort**
- **tar zxvf /home/bubba/Desktop/snortrules-snapshot-CURRENT_s.tar.gz -C /etc/snort**
- **cp etc/* /etc/snort**
- **groupadd snort**
- **useradd -g snort snort**
- **chown snort:snort /var/log/snort**
- **touch /var/log/snort/alert**
- **chown snort:snort /var/log/snort/alert**

- **chmod 600 /var/log/snort/alert**
- **cp /etc/snort/so_rules/precompiled/Ubuntu-10.10.1/i386/2.8.4/*.so /usr/**

local/lib/snort_dynamicrule

- **mv /usr/local/lib/snort_dynamicrule /usr/local/lib/snort_dynamicrules**

The last step is to handle a misspelling.

A.6 Edit snort.conf

The snort.conf file defines how snort will run once the application is started. It is quite long and most items are beyond the scope of this project. We will only be working on a few of the features.

First, open nano (used in this paper) or another text editor with the following command:

- **nano /etc/snort/snort.conf**
- Find the variable RULE_PATH and change to /etc/snort/rules
- Find output and comment out any output modules currently on.
- Find "output log_unified". Insert the following below it: output unified2:

filename snort.log, limit=128

A.7 Set up MySQL

MySQL will serve as the database for the snort application. While not required to run Snort on its own, a database makes it easier to track down events. One short note: this paper does not assume any expertise in database administration, nor does the author claim any significant skill in this area. The commands in this section are sufficient to install and run Snort.

- **mysql -p**

• **create database snort;**

• **grant CREATE, INSERT, SELECT, DELETE, UPDATE on snort.* to**

snort@localhost;

• **SET PASSWORD FOR snort@localhost=PASSWORD('password');**

• **exit**

• **cd /usr/local/snort-2.8.4.1/schemas**

• **mysql -p < create_mysql snort**

Now we will check to see that the Snort database has been correctly installed:

• **mysql -p**

• **SHOW DATABASES;** There should be 2 or more rows as shown in Figure A-1

Server version: 5.1.58-1ubuntu1 (Ubuntu)
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> show databases;
| Database |
| information_schema |
| snort |
2 rows in set (0.00 sec)
mysql>

Figure A-1 MySQL database showing table "snort"

• **USE SNORT;**

• **SHOW TABLES;** There should be 16 rows as shown below in Figure A-2

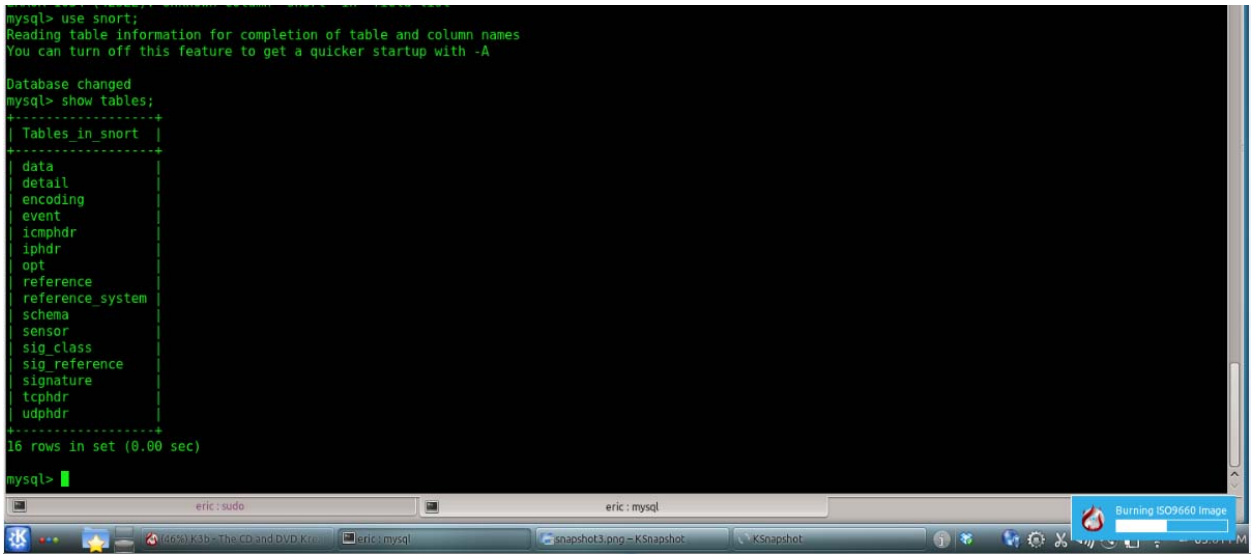

Figure A-2 MySQL Database Table "snort" with 16 rows

A.8 Start Snort.

• In the CLI, type "**snort -c /etc/snort/snort/conf -i eth1"**

• Open a second CLI or new command shell in Konsole (recommended).

• **ls -la /var/log/snort**. Look for 10 digit suffix on snort.log. If there is more than one file, copy the latest one.

Enter the following, then save and exit:

• **/var/log/snort**

• **snort.log**

• <10 digit number from step 2 above>

• 0

A.9 Test Snort

In this step, we will test Snort with a simple rule in the local.rules file. Local rules are rules that the administrator of Snort writes himself and have a convention of starting with SID (Snort ID) of 1,000,000-1,999,999.

- Open a third CLI
- vim /etc/snort/rules/local.rules
- Insert "**alert tcp any any <> any 80 (msg: "Test web activity"; sid:1000001;)**". Save and exit.
- Restart Snort
- Open a web browser
- In the browser, go to any web page.
- Go to *http://localhost/base-1.4.3.1* and look at your events
- If you see a number of events with SID 1000001, Snort works!
- **vim /etc/snort/rules/local.rules** and disable the "Test web activity" rule.

Your Pig is ready to Snort!

Appendix B - Installing Nmap and Zenmap GUI (Attacker PC)

B.1 Installation Overview

The following instructions will guide the user through the installation of the well-known port/vulnerability scanning software Nmap, along with a GUI front-end called Zenmap.

Please Note: All of the following subsections are to be performed on the Attacker PC

B.2 Nmap Installation

The following instructions were taken from http://nmap.org./book/inst-source.html.

- Download the latest version of Nmap in .tar.bz2 (bzip2 compression) or .tgz (gzip compression) format from http://nmap.org/download.html.
- Decompress the downloaded tarball with the following command as:
- **bzip2 -cd nmap-<*VERSION*>.tar.bz2 | tar xvf –**
- With GNU tar, the simpler command **tar xvjf nmap-<*VERSION*>.tar.bz2** does the trick. If you downloaded the .tgz version, replace bzip2 with gzip in the decompression command.
- Change into the newly created directory: **cd nmap-<*VERSION*>**
- Configure the build system: **./configure**
- If the configuration succeeds, an ASCII art dragon appears to congratulate you on successful configuration!

Note: The author was able to use the above instructions to successfully download, install, and use nmap, but they are overly complicated. Here is a much streamlined alternative:

- **apt-get install nmap**

This will install nmap without the commands on the previous page.

B.4 Zenmap Installation

Zenmap as shown in Figure 4-1 is a GUI for Nmap that allows novice users to have "point-and-click" capabilities for selecting which scans they would like to execute, thereby reducing the learning curve. Frequently used scans and results of completed scans can be saved for later use or viewing.

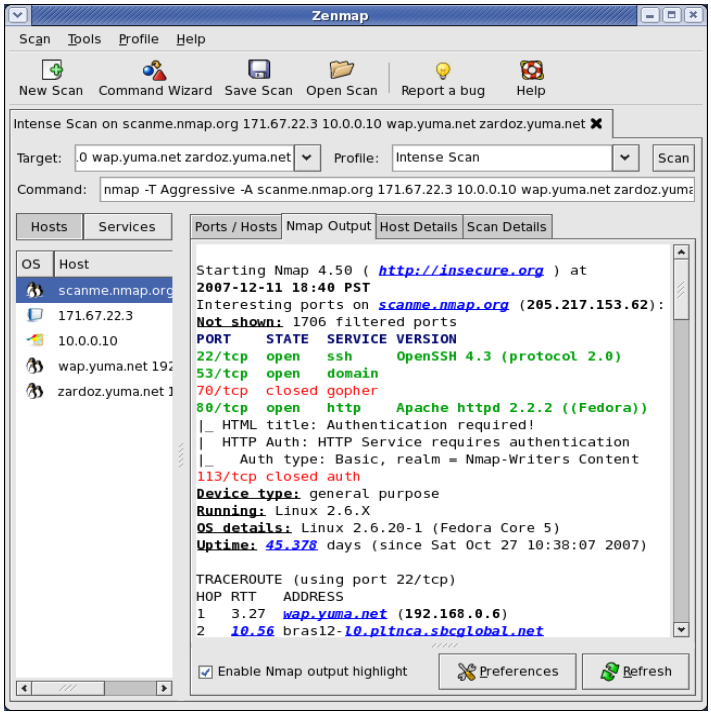
Zenmap
Scan Tools Profile Help
New Scan Command Wizard Save Scan Open Scan Report a bug Help
Intense Scan on scanme.nmap.org 171.67.22.3 10.0.0.10 wap.yuma.net zardoz.yuma.net
Target: .0 wap.yuma.net zardoz.yuma.net Profile: Intense Scan Scan
Command: nmap -T Aggressive -A scanme.nmap.org 171.67.22.3 10.0.0.10 wap.yuma.net zardoz.yuma
Hosts Services
OS Host
scanme.nmap.org
171.67.22.3
10.0.0.10
wap.yuma.net 192
zardoz.yuma.net 1
Ports / Hosts Nmap Output Host Details Scan Details
Starting Nmap 4.50 (http://insecure.org) at
2007-12-11 18:40 PST
Interesting ports on scanme.nmap.org (205.217.153.62):
Not shown: 1706 filtered ports
PORT STATE SERVICE VERSION
22/tcp open ssh OpenSSH 4.3 (protocol 2.0)
53/tcp open domain
70/tcp closed gopher
80/tcp open http Apache httpd 2.2.2 ((Fedora))
|_ HTML title: Authentication required!
| HTTP Auth: HTTP Service requires authentication
|_ Auth type: Basic, realm = Nmap-Writers Content
113/tcp closed auth
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.20-1 (Fedora Core 5)
Uptime: 45.378 days (since Sat Oct 27 10:38:07 2007)
TRACEROUTE (using port 22/tcp)
HOP RTT ADDRESS
1 3.27 wap.yuma.net (192.168.0.6)
2 10.56 bras12-l0.pltnca.sbcglobal.net
Enable Nmap output highlight
Preferences
Refresh

Figure B-1 Zenmap GUI front-end for Nmap

Before installing Zenmap, please ensure Section 4.1 has been completed and Nmap has been installed correctly.

Execute the following

- **apt-get install zenmap**

After the installation has been completed the user may run Zenmap by executing at the CLI:

- **sudo zenmap**

Please note that the previous command MUST be run as a superuser/root. Otherwise, many of Zenmap's functionality is disabled due to insufficient user privileges.

B.5 Executing Port & Vulnerability Scans

Port and vulnerability scans are vital to this project as they verify that SNORT and the SNORT rule sets work properly. There are several types of scans that can be performed

From the simple TCP port scan to the extremely complex and time-consuming "Slow Comprehensive Scan". To execute the scan, simply enter your target's web address / IP address into the "target" field and either select one of the predefined scans from the drop down list to the right or enter your own into the "command" input field.

Appendix C- Conducting Port Scans and Viewing Intrusion Attempts – How to Use This Project

Step C.1 Create Virtual Machines for SNORT and Nmap using Oracle VirtualBox.

- First, go online to https://www.virtualbox.org/wiki/Downloads and download VirtualBox.
- Next, run the downloaded installer and follow the on screen instructions to complete VirtualBox's installation.
- This project requires two (2) virtual Ubuntu Linux machines to work, the next steps will be a walkthrough of the installation process.
- Double-click the VirtualBox icon on the desktop of the Host PC.
- The main program window should appear as shown in Figure C-1

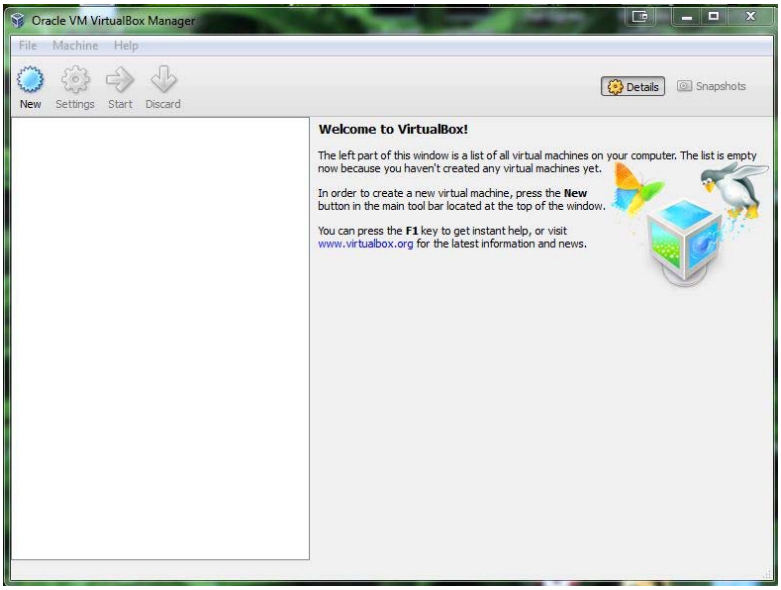
Oracle VM VirtualBox Manager
File
Machine
Help
New
Settings
Start
Discard
Details
Snapshots
Welcome to VirtualBox!
The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.
In order to create a new virtual machine, press the New button in the main tool bar located at the top of the window.
You can press the F1 key to get instant help, or visit www.virtualbox.org for the latest information and news.

Figure C-1 Oracle VirtualBox Main Window

- Click the "**New**" button in the upper left hand side of the window, this starts the creation of the SNORT virtual machine.
- The **"Welcome to the New Virtual Machine Wizard!"** should appear as shown in Figure C-2

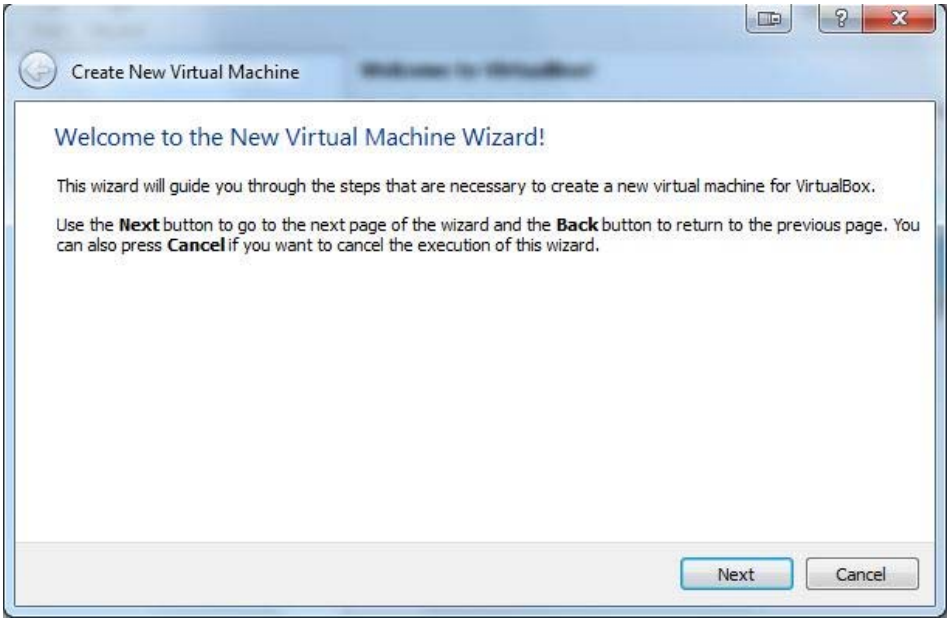
Create New Virtual Machine
Welcome to the New Virtual Machine Wizard!
This wizard will guide you through the steps that are necessary to create a new virtual machine for VirtualBox.
Use the Next button to go to the next page of the wizard and the Back button to return to the previous page. You can also press Cancel if you want to cancel the execution of this wizard.
Next
Cancel

Figure C-2 New Virtual Machine Wizard

- Click **Next**.
- Now enter “Snort” in the **Name** field, Select “Linux” in the **Operating System** drop-down box, and finally select “Ubuntu” (NOT the 64-bit version) in the “Version” drop-down box the “**VM Name and OS Type**”.
- Click **Next**.
- The “**Memory**” window will appear as shown in Figure C-3. The default allocation of 512 MB of RAM for this installation is sufficient.

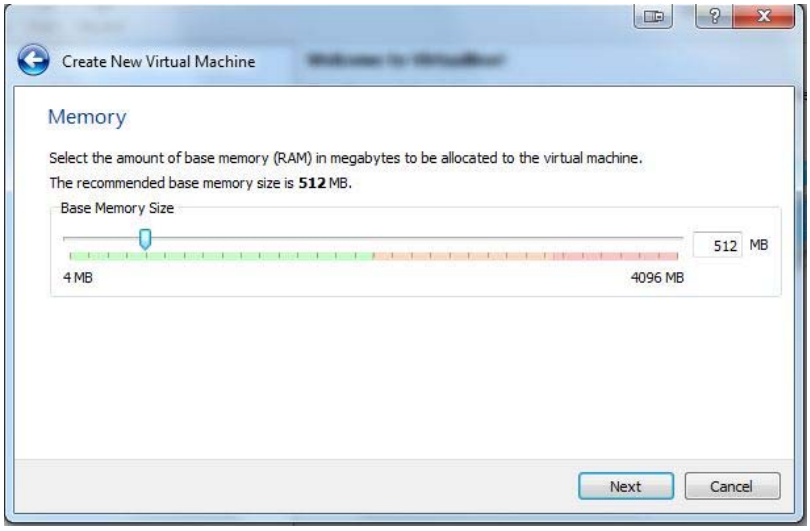
Create New Virtual Machine
Memory
Select the amount of base memory (RAM) in megabytes to be allocated to the virtual machine.
The recommended base memory size is 512 MB.
Base Memory Size
512 MB
4 MB
4096 MB
Next
Cancel

Figure C-3 Memory Allocation Window

- Click **Next**.
- The virtual hard disk images (.vdi) files have already been made. As shown in Figure C-4 click "**Use Existing Hard Disk"** and click the folder icon to the right to browse for the SNORT .vdi file needed.

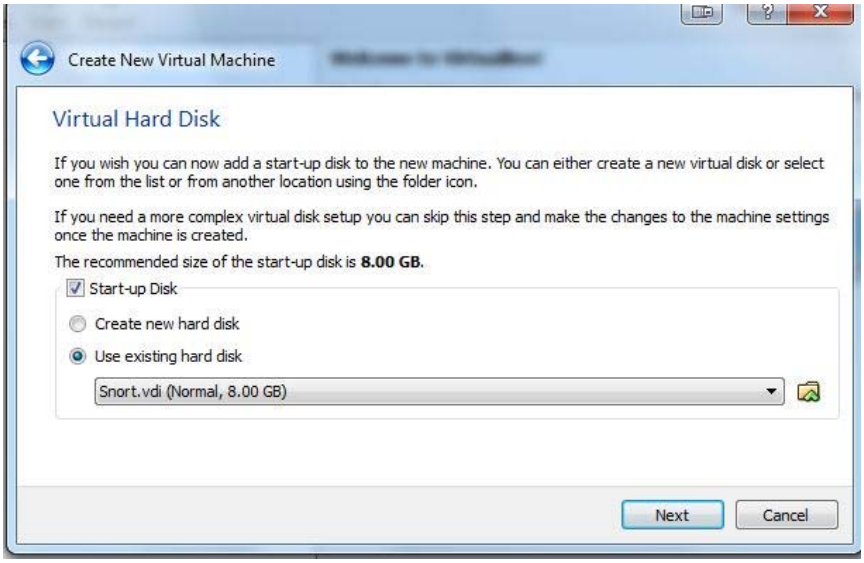
Create New Virtual Machine
Virtual Hard Disk
If you wish you can now add a start-up disk to the new machine. You can either create a new virtual disk or select one from the list or from another location using the folder icon.
If you need a more complex virtual disk setup you can skip this step and make the changes to the machine settings once the machine is created.
The recommended size of the start-up disk is 8.00 GB.
Start-up Disk
Create new hard disk
Use existing hard disk
Snort.vdi (Normal, 8.00 GB)
Next
Cancel

Figure C-4 Virtual Hard Disk Selection/Creation Window

- Navigate the file manager/explorer to the currently logged in user's folder.

- As shown in figure C-5, The folder that contains the .vdi file for SNORT resides in **C:\Users\<*insert your user name here*>\VirtualBox VMs\Snort**

  ****Omit the left and right carats!****

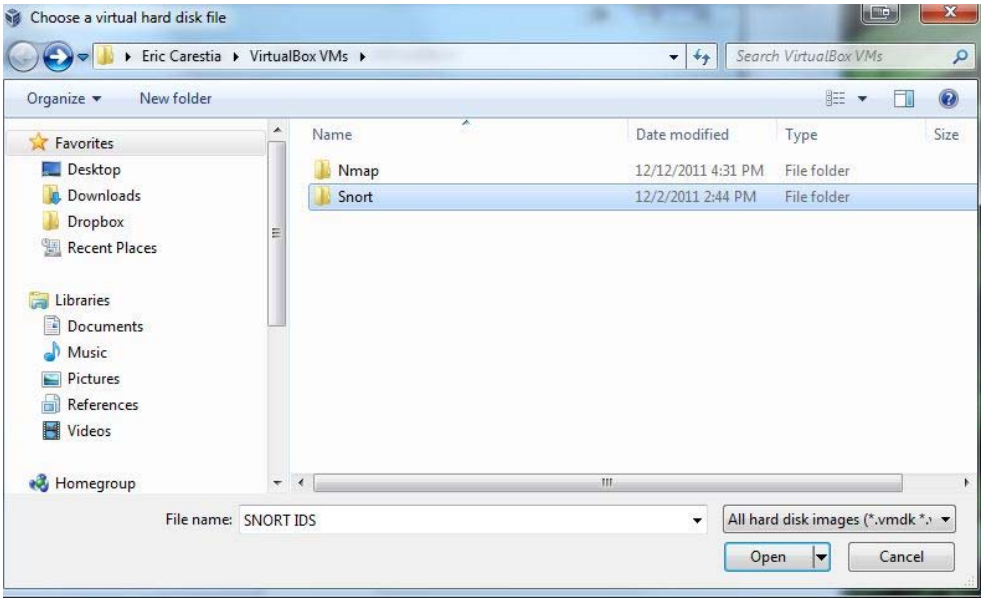

Figure C-5 File Explorer (Host PC) with Snort Directory Selected

- Open the "Snort" folder and select **Snort.vdi** as shown in Figure C-6.

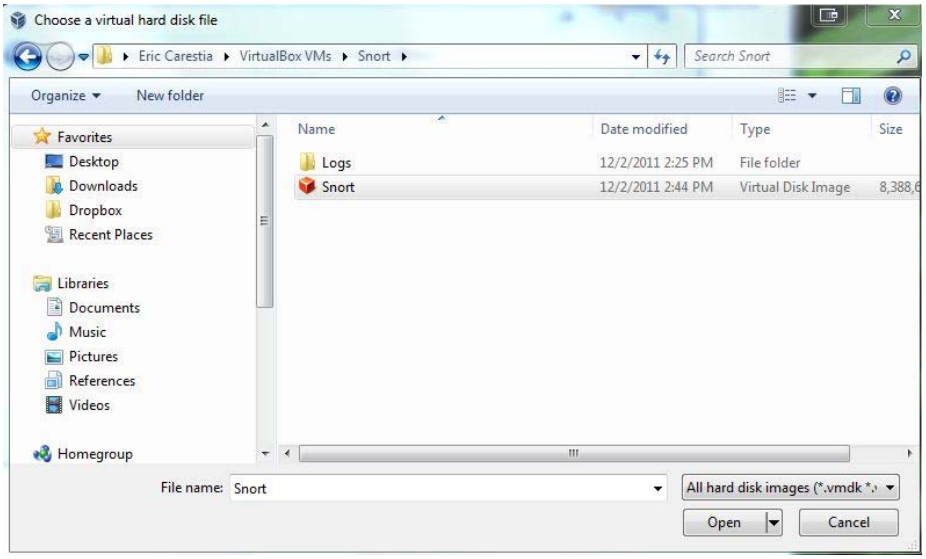

Figure C-6 Snort.vdi Selection

- Once loaded, the **"Summary"** window will appear. This window gives an overview of the Name, Operating System, Base Memory, and Virtual Hard Disk specifications.
- Click **Create**
- The new SNORT virtual machine will now appear in the main window of Oracle VirtualBox as shown in Figure C-7.

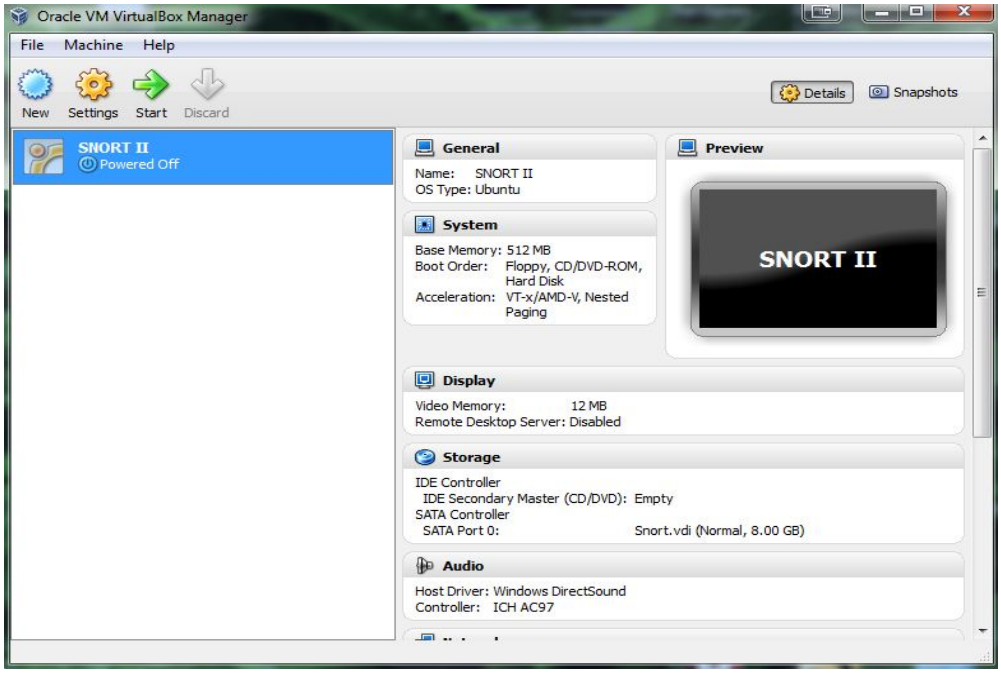

Figure C-7 SNORT Virtual Machine Successfully Created.

- To create the virtual machine for the NMap Attacker, repeat Step C-1 and replace every instance of "**SNORT"** with "**NMAP"**.

Step C.2 Start SNORT Intrusion Detection System

- Open VirtualBox and start the SNORT IDS virtual machine.
- Once they have been loaded, login to the account snort on both machines with the following username: **snort** and password: **snort**.

- In the SNORT IDS environment, open a command terminal and enter the following as shown in Figure C-1

  sudo snort –i eth0 –c /usr/local/snort/etc/snort.conf

- Press Enter

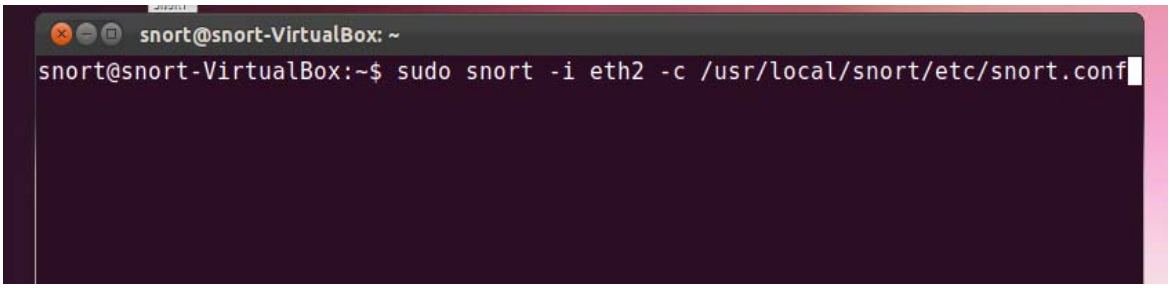

Figure C-1 Command to Start Snort in Intrusion Detection Mode

- Once Snort has finished initializing, a message at the bottom of the command window should appear that says **"Initialization Complete"** as shown in Figure C-2
- Open a new command window and type **ifconfig** and then press Enter.
- IMPORTANT: Write down the IP address for the Ethernet adapter (in this case, eth0) it is needed for the attacker to scan the Target PC!

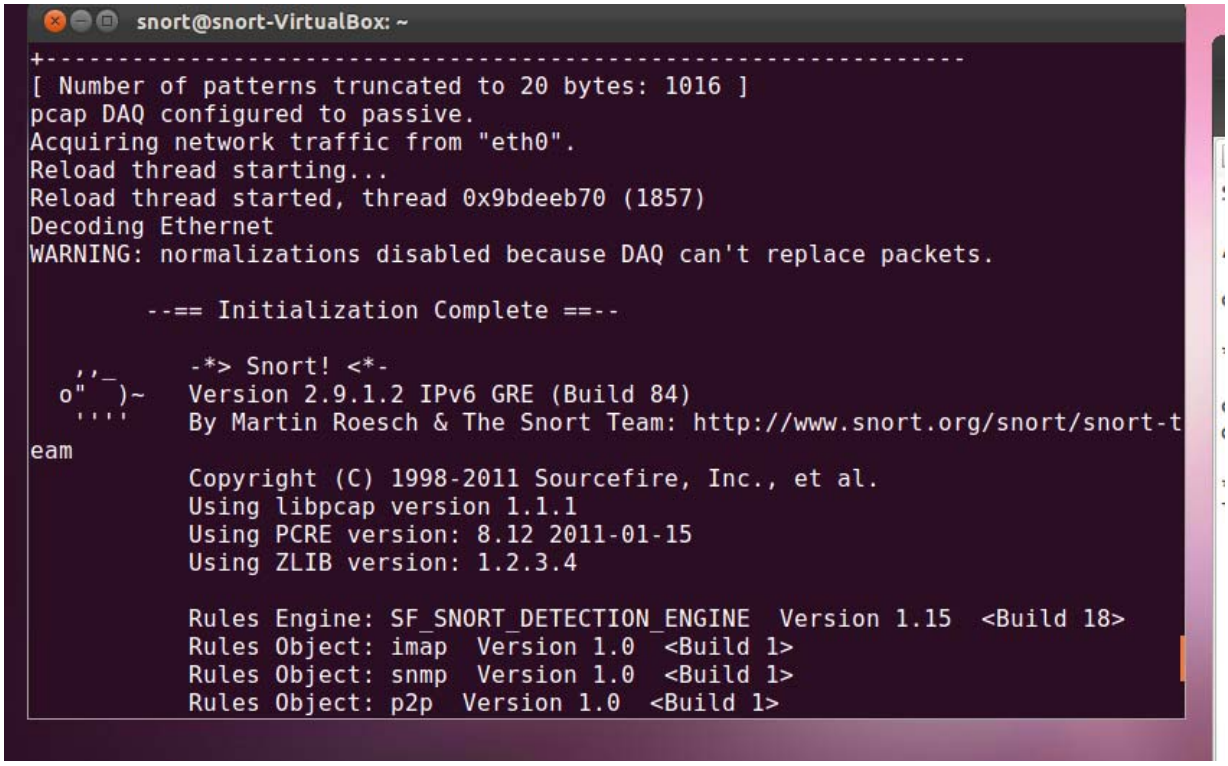
snort@snort-VirtualBox: ~
[Number of patterns truncated to 20 bytes: 1016]
pcap DAQ configured to passive.
Acquiring network traffic from "eth0".
Reload thread starting...
Reload thread started, thread 0x9bdeeb70 (1857)
Decoding Ethernet
WARNING: normalizations disabled because DAQ can't replace packets.
--== Initialization Complete ==--
-*> Snort! <*-
Version 2.9.1.2 IPv6 GRE (Build 84)
By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort-t
eam
Copyright (C) 1998-2011 Sourcefire, Inc., et al.
Using libpcap version 1.1.1
Using PCRE version: 8.12 2011-01-15
Using ZLIB version: 1.2.3.4
Rules Engine: SF_SNORT_DETECTION_ENGINE Version 1.15 <Build 18>
Rules Object: imap Version 1.0 <Build 1>
Rules Object: snmp Version 1.0 <Build 1>
Rules Object: p2p Version 1.0 <Build 1>

Figure C-2 Snort successfully initialized

Step C-2 Conduct NMap port scan of Target PC

- Return to the Oracle VirtualBox main window and run the Nmap virtual machine.
- Once loaded, open a command window and type the following as shown in Figure C-3:

  sudo zenmap

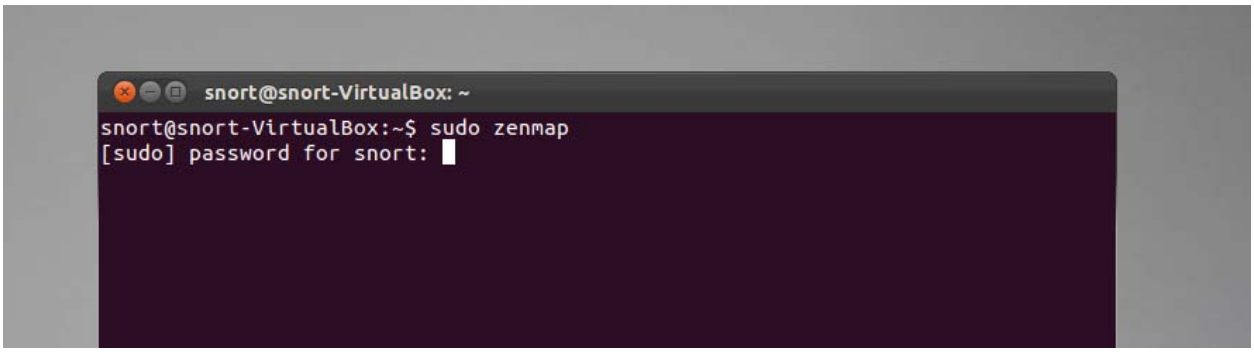

Figure C-3 Start Zenmap, the NMap GUI front end

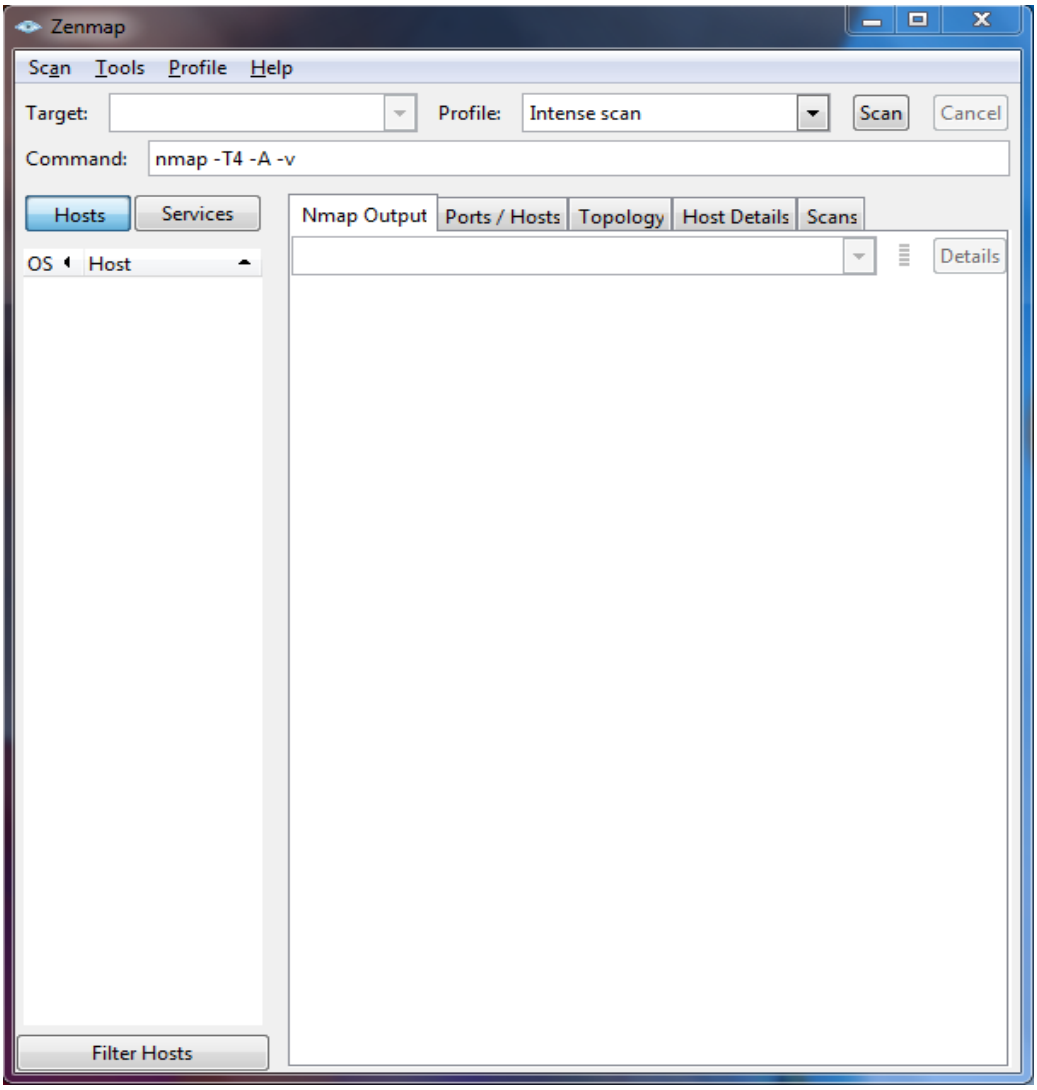

Figure C-4 ZenMap Program

- Once Zenmap initializes, the program window should appear as shown in Figure C-4.
- There are many predefined scans one can choose, but we will chose “Intense Scan All TCP ports” from the field labeled **Profile**’s drop down menu shown in Figure C-5.
- Next, enter the IP address of the Target PC into the field labeled Target as shown in Figure C-5 and click **Scan**.

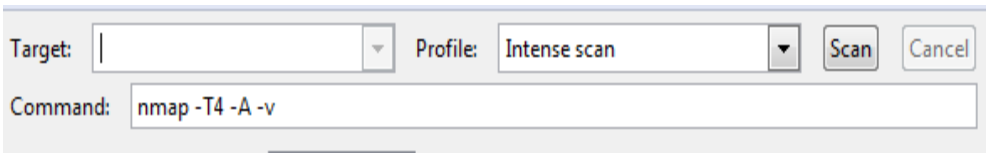

Figure C-5 Target and Profile Scan Selection

- Once the scan completes, view the results in the **NMap Output** tab, one should see something similar to Figure C-6.

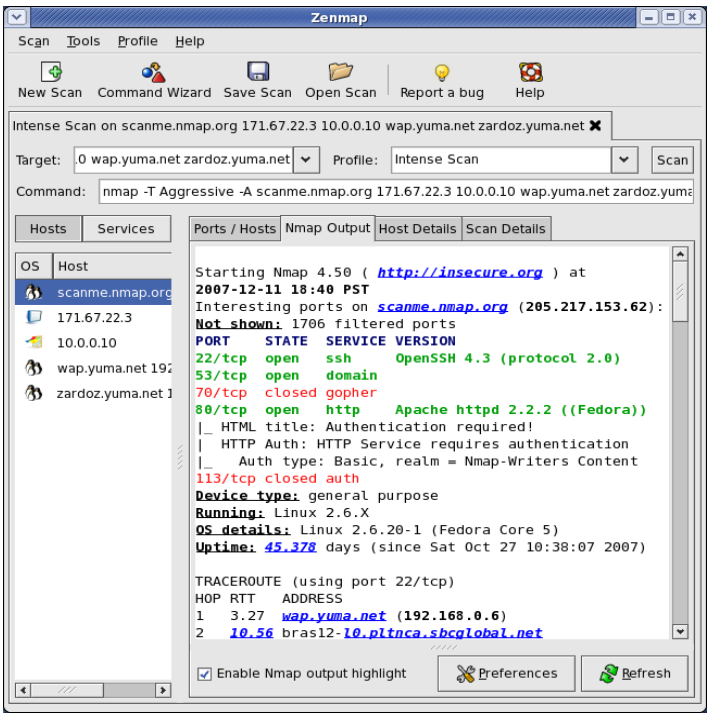

Figure C-6 NMap Output Results

Step C-3 Viewing Intrusion Attempts (Target PC)

Viewing the intrusion attempts made by unauthorized users is at the core of the SNORT IDS software. This project uses a MySQL database to store the events that SNORT generates from intrusion attempts.

To view the access the snort database directly, execute the following:

- Open a new command line window (gnome-terminal or Konsole)
- Execute **mysql –u snort –p**
- Enter your password for the user “snort” which is “snort”
- Enter the following query: **show databases;**

If entered correctly, the output displayed should match Figure C-7.

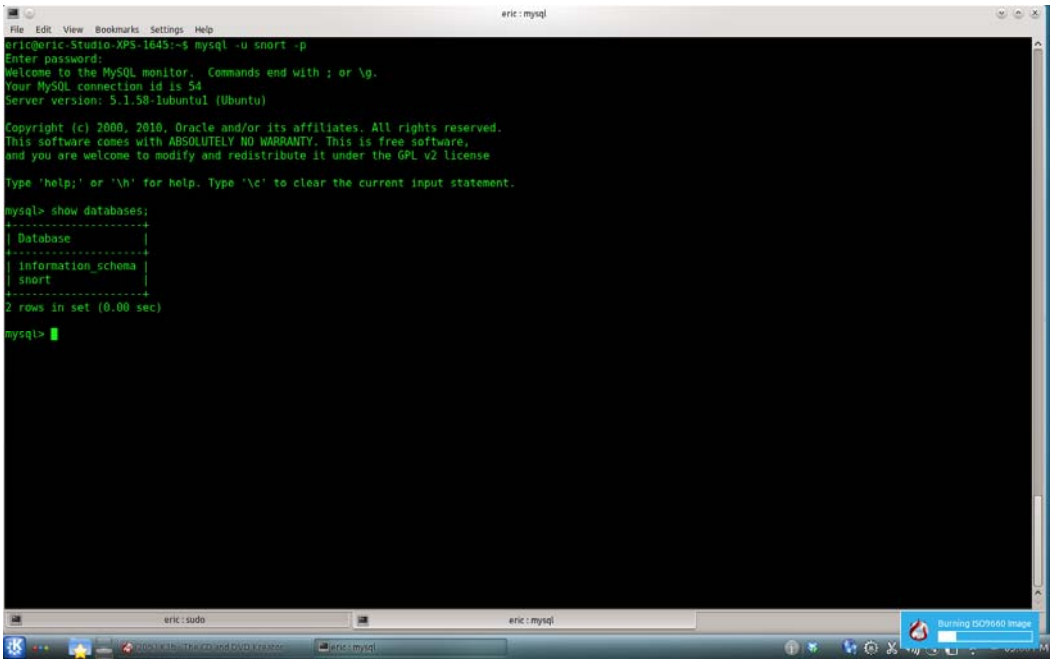
eric : mysql
File Edit View Bookmarks Settings Help
eric@eric-Studio-XPS-1645:~$ mysql -u snort -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 54
Server version: 5.1.58-1ubuntu1 (Ubuntu)
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
| snort |
+--------------------+
2 rows in set (0.00 sec)
mysql>
eric : sudo
eric : mysql
Burning ISO9660 Image

Figure C-7 MySQL Console showing “snort” database

The data generated from the intrusion attempts is stored in the table created named “snort”

- Execute the following query: **use snort;**
- Execute the following query: **show tables;**

As shown in Figure C-8, there should be 16 rows within the “snort” database.

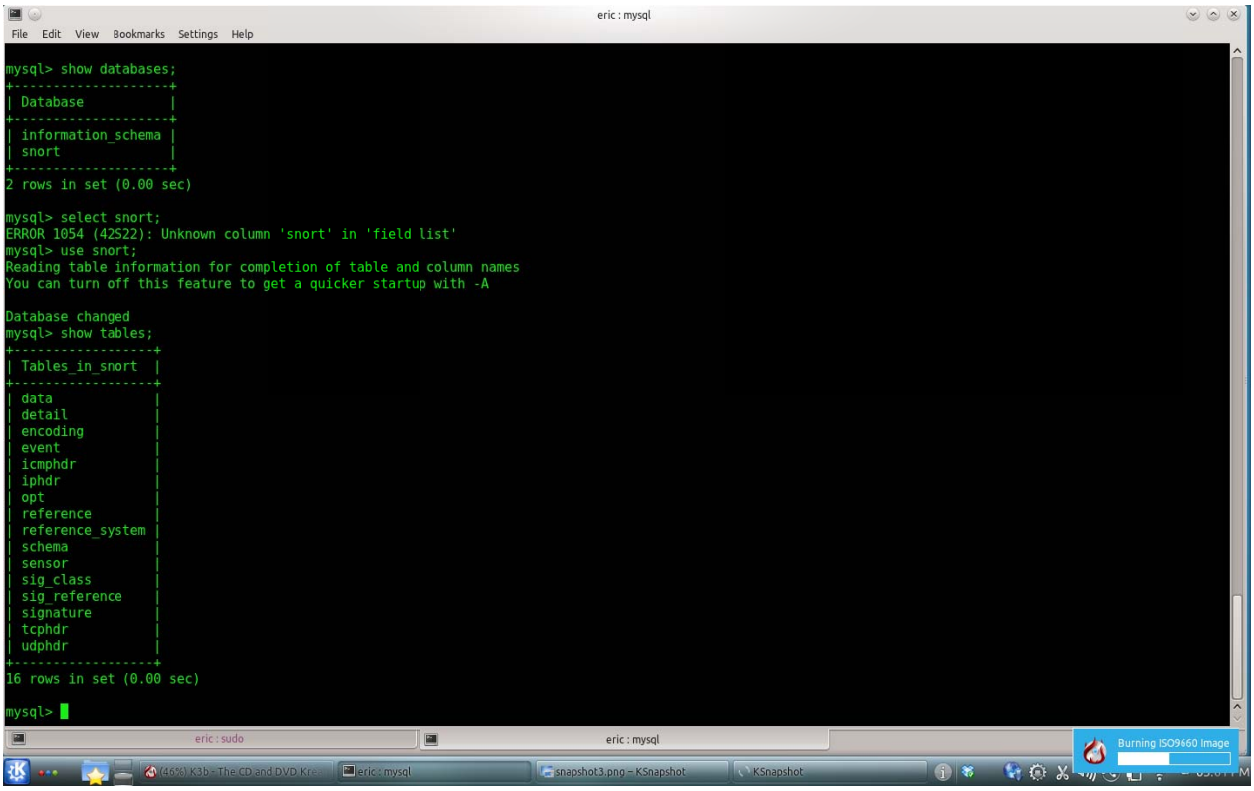

Figure C-8 MySQL Console Showing tables in "snort" database

To view Intrusion attempts, execute the following

- **SELECT * FROM signature, event;**
- Press Enter;
- Once query is finished executing, the output should resemble Figure C-9.

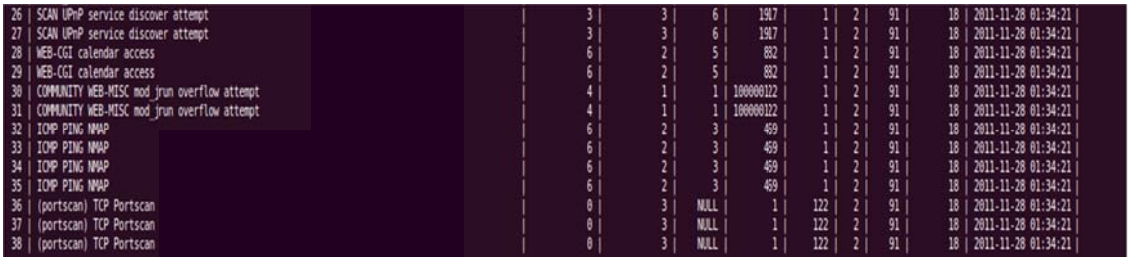

Figure C-9 Intrusion Attempts Logged into MySQL Database Snort.

The query results should display numerous entries that have a signature indicating that a particular entry is of a NMap ping or port scan made against the Target PC at the given date and time.